Network security - Firewall Exploration Lab

Environment Setup Using Containers

Container Setup and Commands

Download the Ubuntu 20.04 VM

Ubuntu 20.04 VM

If you prefer to create a SEED VM on your local computers, there are two ways to do that: (1) use a pre-built SEED VM; (2) create a SEED VM from scratch.

Approach 1: Use a pre-built SEED VM. We provide a pre-built SEED Ubuntu 20.04 VirtualBox image (`SEED-Ubuntu20.04.zip`, size: 4.0 GB), which can be downloaded from the following links.

- Google Drive
- DigitalOcean
- MD5 value: f3d2227c92219265679400064a0a1287
- VM Manual: follow this manual to install the VM on your computer

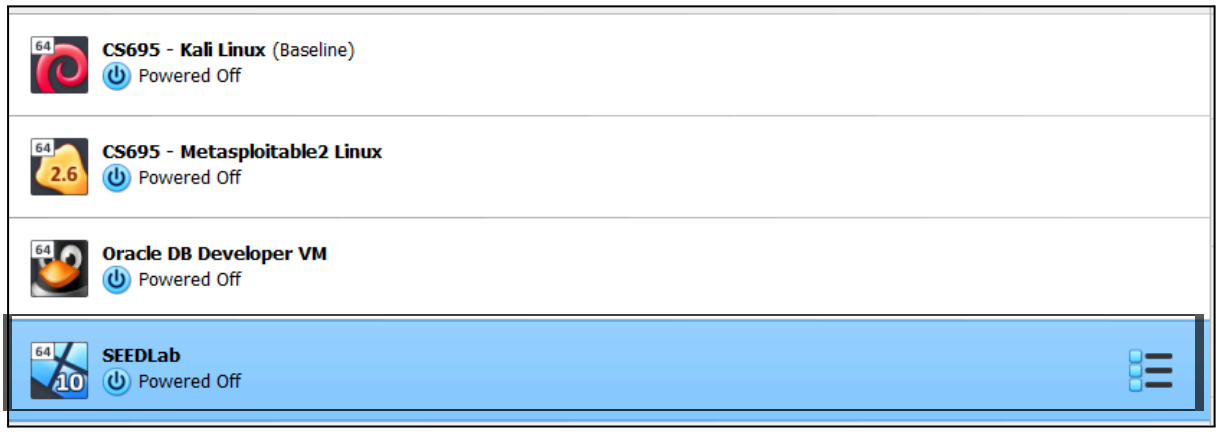
Approach 2: Build a SEED VM from scratch. The procedure to build the SEED VM used in Approach 1 is fully documented, and the code is open source. If you want to build your own SEED Ubuntu VM from scratch, you can use the following manual.

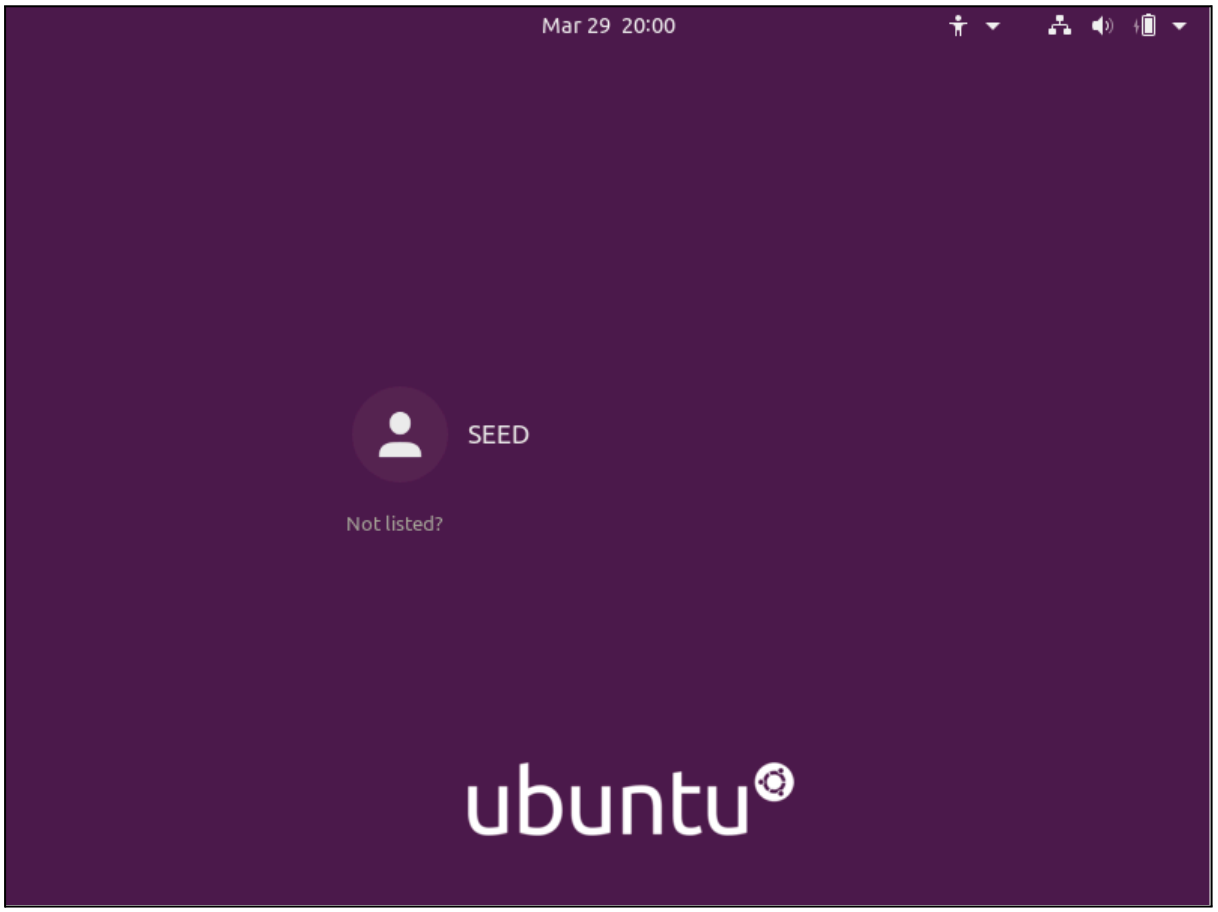
- How to build a SEED VM from scratch

Extract the downloaded zip file

```
PS D:\VM Images\CS690 - Seedlab1> dir


    Directory: D:\VM Images\CS690 - Seedlab1


Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
d-----         3/28/2023  12:34 AM                Labsetup
-a----         3/27/2023  11:41 PM           4555 Labsetup.zip
-a----         3/28/2023   1:08 AM     3595552981 SEEDUbuntu-16.04-32bit.zip


PS D:\VM Images\CS690 - Seedlab1> Expand-Archive -Path SEEDUbuntu-16.04-32bit.zip -DestinationPath SEEDUbuntu
PS D:\VM Images\CS690 - Seedlab1> dir


    Directory: D:\VM Images\CS690 - Seedlab1


Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
d-----         3/28/2023  12:34 AM                Labsetup
d-----         3/28/2023   1:12 AM                SEEDUbuntu
-a----         3/27/2023  11:41 PM           4555 Labsetup.zip
-a----         3/28/2023   1:08 AM     3595552981 SEEDUbuntu-16.04-32bit.zip


PS D:\VM Images\CS690 - Seedlab1>
```

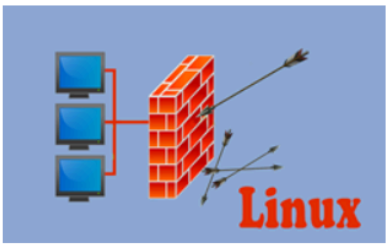
Load the Virtual disk image into Virtual Box





Inside VM - Download the *Labsetup.zip* and extract it

Overview



The learning objective of this lab is two-fold: learning how firewalls work, and setting up a simple firewall for a network. Students will first implement a simple stateless packet-filtering firewall, which inspects packets, and decides whether to drop or forward a packet based on firewall rules. Through this implementation task, students can get the basic ideas on how firewall works.

Linux already has a built-in firewall, also based on netfilter. This firewall is called iptables. Students will be given a simple network topology, and are asked to use iptables to set up firewall rules to protect the network. Students will also be exposed to several other interesting applications of iptables.

Tasks (PDF)

- **VM version:** This lab has been tested on our SEED Ubuntu-20.04 VM
- **Lab setup files:**: Labsetup.zip
- **Manual:**: Docker manual

Extract the zip file

```
[03/28/23]seed@VM:~$ cd Downloads
[03/28/23]seed@VM:~/Downloads$ ls -lrt
total 8
-rw-rw-r-- 1 seed seed 4555 Mar 28 18:24 Labsetup.zip
[03/28/23]seed@VM:~/Downloads$ unzip Labsetup.zip
Archive:  Labsetup.zip
   creating: Labsetup/
   creating: Labsetup/router/
  inflating: Labsetup/router/Dockerfile
   creating: Labsetup/Files/
   creating: Labsetup/Files/kernel_module/
  inflating: Labsetup/Files/kernel_module/Makefile
  inflating: Labsetup/Files/kernel_module/hello.c
   creating: Labsetup/Files/packet_filter/
  inflating: Labsetup/Files/packet_filter/Makefile
  inflating: Labsetup/Files/packet_filter/seedFilter.c
  inflating: Labsetup/Files/packet_filter/.gitignore
  inflating: Labsetup/docker-compose.yml
   creating: Labsetup/volumes/
 extracting: Labsetup/volumes/.gitignore
[03/28/23]seed@VM:~/Downloads$ ls -lrt
total 12
drwxrwxr-x 5 seed seed 4096 Nov  4  2021 Labsetup
-rw-rw-r-- 1 seed seed 4555 Mar 28 18:24 Labsetup.zip
```

Contents of the folder

```
[03/28/23]seed@VM:~/Downloads$ cd Labsetup
[03/28/23]seed@VM:~/.../Labsetup$ ls -lrt
total 16
drwxrwxr-x 4 seed seed 4096 Jan 15  2021 Files
drwxrwxr-x 2 seed seed 4096 Jan 15  2021 volumes
drwxrwxr-x 2 seed seed 4096 Oct 22  2021 router
-rw-rw-r-- 1 seed seed 3014 Nov  4  2021 docker-compose.yml
```

Now, build the services in the VM using the file provided *docker-compose.yml* using command - '*dcbuild*'

dcbuild helps to build or rebuild the services

```
[03/29/23]seed@VM:~/.../Labsetup$ dcbuild
HostA uses an image, skipping
Host1 uses an image, skipping
Host2 uses an image, skipping
Host3 uses an image, skipping
Building Router
Step 1/2 : FROM handsonsecurity/seed-ubuntu:large
large: Pulling from handsonsecurity/seed-ubuntu
da7391352a9b: Pulling fs layer
14428a6d4bcd: Downloading [==========================================>      ] 14428a6d4bcd: Downloading [==================================
===============>] da7391352a9b: Downloading [>                                                  ] da7391352a9b: Downloading [=>
                                ]    884kB/28.56MBwnload complete
da7391352a9b: Downloading [=======================================>           ]  21.82MB/28.56MBiting
da7391352a9b: Pull complete
14428a6d4bcd: Pull complete
2c2d948710f2: Pull complete
b5e99359ad22: Pull complete
3d2251ac1552: Pull complete
1059cf087055: Pull complete
b2afee800091: Pull complete
c2ff2446bab7: Pull complete
4c584b5784bd: Pull complete
Digest: sha256:41efab02008f016a7936d9cadfbe8238146d07c1c12b39cd63c3e73a0297c07a
Status: Downloaded newer image for handsonsecurity/seed-ubuntu:large
 ---> cecb04fbf1dd
Step 2/2 : RUN apt-get update     && apt-get install -y kmod      && apt-get clean
 ---> Running in 248c7304206e
Get:1 http://security.ubuntu.com/ubuntu focal-security InRelease [114 kB]
Get:2 http://archive.ubuntu.com/ubuntu focal InRelease [265 kB]
Get:3 http://security.ubuntu.com/ubuntu focal-security/restricted amd64 Packages [2060 kB]
Get:4 http://archive.ubuntu.com/ubuntu focal-updates InRelease [114 kB]
Get:5 http://archive.ubuntu.com/ubuntu focal-backports InRelease [108 kB]
Get:6 http://archive.ubuntu.com/ubuntu focal/restricted amd64 Packages [33.4 kB]
Get:7 http://archive.ubuntu.com/ubuntu focal/main amd64 Packages [1275 kB]
Get:8 http://security.ubuntu.com/ubuntu focal-security/multiverse amd64 Packages [28.5 kB]
Get:9 http://security.ubuntu.com/ubuntu focal-security/universe amd64 Packages [1027 kB]
```

Now, startup the built services using docker command - '*dcup*'

```
[03/29/23]seed@VM:~/.../Labsetup$ dcup
Creating network "net-10.9.0.0" with the default driver
Creating network "net-192.168.60.0" with the default driver
Creating hostA-10.9.0.5     ... done
Creating seed-router        ... done
Creating host2-192.168.60.6 ... done
Creating host3-192.168.60.7 ... done
Creating host1-192.168.60.5 ... done
Attaching to host3-192.168.60.7, hostA-10.9.0.5, host2-192.168.60.6, host1-192.168.60.5, seed-router
hostA-10.9.0.5 |  * Starting internet superserver inetd                 [ OK ]
host1-192.168.60.5 |  * Starting internet superserver inetd             [ OK ]
seed-router |  * Starting internet superserver inetd                    [ OK ]
host3-192.168.60.7 |  * Starting internet superserver inetd             [ OK ]
host2-192.168.60.6 |  * Starting internet superserver inetd             [ OK ]
```

Now, verify the running and stopped containers by using a docker command - '*dockps*' (in new tab of the terminal, because the current tab has the services running up'

```
[03/29/23]seed@VM:~/.../Labsetup$ dockps
06d66a6c664d  host1-192.168.60.5
f78e15e8c54a  host3-192.168.60.7
f863f0cb47dd  seed-router
ff64c9ae7dae  host2-192.168.60.6
56f473f43aa8  hostA-10.9.0.5
```

Task 1: Implementing a Simple Firewall

Task 1.A: Implement a Simple Kernel Module

Files available in *kernel_module* directory under *Files* directory

```
[03/29/23]seed@VM:~/.../Files$ cd kernel_module
[03/29/23]seed@VM:~/.../kernel_module$ ls -lrt
total 8
-rw-rw-r-- 1 seed seed 156 Jan 13  2021 Makefile
-rw-rw-r-- 1 seed seed 279 Mar 14  2022 hello.c
```

To implement a simple kernel module, from the *Labsetup* folder, under Files>kernel_modules directory we have 2 special files that can implement a simple kernel module

- hello.c - has the setup to implement a simple kernel module
- Makefile - is an executable file, that will generate Loadable Kernel Module

Compile *hello.c* file into loadable kernel module *hello.ko*

```
[03/29/23]seed@VM:~/.../kernel_module$ make
make -C /lib/modules/5.4.0-54-generic/build M=/home/seed/Downloads/Labsetup/Files/kernel_module modules
make[1]: Entering directory '/usr/src/linux-headers-5.4.0-54-generic'
  CC [M]  /home/seed/Downloads/Labsetup/Files/kernel_module/hello.o
  Building modules, stage 2.
  MODPOST 1 modules
  CC [M]  /home/seed/Downloads/Labsetup/Files/kernel_module/hello.mod.o
  LD [M]  /home/seed/Downloads/Labsetup/Files/kernel_module/hello.ko
make[1]: Leaving directory '/usr/src/linux-headers-5.4.0-54-generic'
```

List of files

```
[03/29/23]seed@VM:~/.../kernel_module$ ls -lrt
total 32
-rw-rw-r-- 1 seed seed  156 Jan 13  2021 Makefile
-rw-rw-r-- 1 seed seed  279 Mar 14  2022 hello.c
-rw-rw-r-- 1 seed seed 2048 Mar 29 20:20 hello.o
-rw-rw-r-- 1 seed seed   59 Mar 29 20:20 hello.mod
-rw-rw-r-- 1 seed seed   59 Mar 29 20:20 modules.order
-rw-rw-r-- 1 seed seed    0 Mar 29 20:20 Module.symvers
-rw-rw-r-- 1 seed seed  560 Mar 29 20:20 hello.mod.c
-rw-rw-r-- 1 seed seed 2800 Mar 29 20:20 hello.mod.o
-rw-rw-r-- 1 seed seed 3968 Mar 29 20:20 hello.ko
```

Load the module (inserting module), List module

```
[03/29/23]seed@VM:~/.../kernel_module$ sudo insmod hello.ko
[03/29/23]seed@VM:~/.../kernel_module$ lsmod | grep hello
hello                  16384  0
```

With the help of the *insmod* command, we can insert the Loadable kernel modules into the kernel. Here, the hello kernel module successfully loaded to the kernel

Check the messages

```
[03/29/23]seed@VM:~/.../kernel_module$ dmesg
[ 1960.128664] Hello World!
```

As from the code, if the module is inserted successfully into the kernel *Hello World!* Message from the syslog file should display. The message we can see using the command *dmesg*

Remove the module

```
[03/29/23]seed@VM:~/.../kernel_module$ sudo rmmod hello.ko
[03/29/23]seed@VM:~/.../kernel_module$ lsmod | grep hello
[03/29/23]seed@VM:~/.../kernel_module$
```

With the use of *rmmod* command, i removed the *hello* kernel module successfully from the kernel

Check the messages

```
[03/29/23]seed@VM:~/.../kernel_module$ dmesg
[ 1960.128664] Hello World!
[ 1981.974060] Bye-bye World!.
[03/29/23]seed@VM:~/.../kernel_module$
```

Once the module was removed successfully, again *Bye-bye World!* Message displayed

Task 1.B: Implement a Simple Firewall Using Netfilter

Contents of the *packet_filter* directory

```
[03/29/23]seed@VM:~/.../Files$ cd packet_filter
[03/29/23]seed@VM:~/.../packet_filter$ ls -lrt
total 8
-rw-rw-r-- 1 seed seed 2746 Jan 13  2021 seedFilter.c
-rw-rw-r-- 1 seed seed  236 Jan 13  2021 Makefile
[03/29/23]seed@VM:~/.../packet_filter$
```

To implement a simple firewall, from the *Labsetup* folder, we have 2 special files that can implement a firewall

- seedFilter.c - has all the setup to implement a simple firewall

- Makefile - is an executable file, that will generate Loadable Kernel Module

Tasks

Before going through further, there is a mention of the *icmp* protocol header in the instructions manual. So , I included the linux header in the *seedFilter.c* file.

```
#include <linux/kernel.h>
#include <linux/module.h>
#include <linux/netfilter.h>
#include <linux/netfilter_ipv4.h>
#include <linux/ip.h>
#include <linux/tcp.h>
#include <linux/udp.h>
#include <linux/icmp.h>
#include <linux/if_ether.h>
#include <linux/inet.h>
```

Task 1.

Compile *seedFilter.c* file into loadable kernel module *seedFilter.ko*

```
[03/29/23]seed@VM:~/.../packet_filter$ make
make -C /lib/modules/5.4.0-54-generic/build M=/home/seed/Downloads/Labsetup/Files/packet_filter modules
make[1]: Entering directory '/usr/src/linux-headers-5.4.0-54-generic'
  CC [M]  /home/seed/Downloads/Labsetup/Files/packet_filter/seedFilter.o
  Building modules, stage 2.
  MODPOST 1 modules
  CC [M]  /home/seed/Downloads/Labsetup/Files/packet_filter/seedFilter.mod.o
  LD [M]  /home/seed/Downloads/Labsetup/Files/packet_filter/seedFilter.ko
make[1]: Leaving directory '/usr/src/linux-headers-5.4.0-54-generic'
```

With the help of *make* command, the Makefile will execute and generates the Loadable kernel module

```
[03/29/23]seed@VM:~/.../packet_filter$ ls -lrt
total 40
-rw-rw-r-- 1 seed seed 2746 Jan 13  2021 seedFilter.c
-rw-rw-r-- 1 seed seed  236 Jan 13  2021 Makefile
-rw-rw-r-- 1 seed seed 5160 Mar 29 21:15 seedFilter.o
-rw-rw-r-- 1 seed seed   64 Mar 29 21:15 seedFilter.mod
-rw-rw-r-- 1 seed seed   64 Mar 29 21:15 modules.order
-rw-rw-r-- 1 seed seed  560 Mar 29 21:15 seedFilter.mod.c
-rw-rw-r-- 1 seed seed    0 Mar 29 21:15 Module.symvers
-rw-rw-r-- 1 seed seed 2808 Mar 29 21:15 seedFilter.mod.o
-rw-rw-r-- 1 seed seed 7088 Mar 29 21:15 seedFilter.ko
[03/29/23]seed@VM:~/.../packet_filter$
```

Load the module (inserting module), List module

```
[03/29/23]seed@VM:~/.../packet_filter$ sudo insmod seedFilter.ko
[03/29/23]seed@VM:~/.../packet_filter$ lsmod | grep seedFilter
seedFilter             16384  0
```

With the help of *insmod* (insert module) command we can insert a Loadable kernel module into the kernel. Here we inserted the module successfully into the kernel.

Check the messages

```
[03/29/23]seed@VM:~/.../packet_filter$ dmesg
[ 1960.128664] Hello World!
[ 1981.974060] Bye-bye World!.
[ 4822.743109] Registering filters.
[ 4826.278493] *** LOCAL_OUT
[ 4826.278495]     10.0.2.15  --> 128.197.253.126 (UDP)
[ 4826.283467] *** LOCAL_OUT
[ 4826.283469]     10.0.2.15  --> 91.189.91.48 (TCP)
[ 4826.286846] *** LOCAL_OUT
[ 4826.286848]     10.0.2.15  --> 91.189.91.48 (TCP)
[ 4826.287060] *** LOCAL_OUT
[ 4826.287061]     10.0.2.15  --> 91.189.91.48 (TCP)
[ 4826.298258] *** LOCAL_OUT
[ 4826.298270]     10.0.2.15  --> 91.189.91.48 (TCP)
[ 4826.298582] *** LOCAL_OUT
[ 4826.298584]     10.0.2.15  --> 91.189.91.48 (TCP)
[ 4878.068939] *** LOCAL_OUT
[ 4878.068941]     10.0.2.15  --> 35.161.94.248 (TCP)
```

Once the module is successfully inserted, the filters will start registering into the kernel for the firewall.

Demonstration

```
[03/29/23]seed@VM:~/.../packet_filter$ dig @8.8.8.8 www.example.com

; <<>> DiG 9.16.1-Ubuntu <<>> @8.8.8.8 www.example.com
; (1 server found)
;; global options: +cmd
;; connection timed out; no servers could be reached
```

To test the filters registered, *dig @8.8.8.8 www.example.com* is executed which will query the DNS server of google which is 8.8.8.8 for www.example.com. But because of blocking the IP address, it is unreachable.

Task 2.

Removing all the make files

```
[03/29/23]seed@VM:~/.../packet_filter$ make clean
make -C /lib/modules/5.4.0-54-generic/build M=/home/seed/Downloads/Labsetup/Files/packet_filter clean
make[1]: Entering directory '/usr/src/linux-headers-5.4.0-54-generic'
  CLEAN   /home/seed/Downloads/Labsetup/Files/packet_filter/Module.symvers
make[1]: Leaving directory '/usr/src/linux-headers-5.4.0-54-generic'
```

Before continuing further, it is good to remove the previously generated makefiles using *make clean* command to avoid discrepancies.

```
[03/29/23]seed@VM:~/.../packet_filter$ ls -lrt
total 8
-rw-rw-r-- 1 seed seed  236 Jan 13  2021 Makefile
-rw-rw-r-- 1 seed seed 2770 Mar 29 21:22 seedFilter.c
```

After removing the makefile generated files, we have the original files.

Create a copy of the *seedfilter.c,* so the original file will not get disturbed by the changes made.

```
[03/29/23]seed@VM:~/.../packet_filter$ cp seedFilter.c seedFilter_1.c
[03/29/23]seed@VM:~/.../packet_filter$ ls -lrt
total 12
-rw-rw-r-- 1 seed seed  236 Jan 13  2021 Makefile
-rw-rw-r-- 1 seed seed 2770 Mar 29 21:22 seedFilter.c
-rw-rw-r-- 1 seed seed 2770 Mar 29 21:39 seedFilter_1.c
```

To avoid confusion, it is better to create a copy of the original file.

Change the file name in *Makefile*

```
#obj-m += seedFilter.o
obj-m += seedFilter_1.o
all:
	make -C /lib/modules/$(shell uname -r)/build M=$(PWD) modules

clean:
	make -C /lib/modules/$(shell uname -r)/build M=$(PWD) clean

ins:
	sudo dmesg -C
	sudo insmod seedFilter.ko

rm:
	sudo rmmod seedFilter
```

I edited the *Makefile* by including the *seedFilter_1.o* to execute instead of *seedFilter.o*

Now, declared 3 additional filters for the hooks

```
static struct nf_hook_ops hook1, hook2, hook3, hook4, hook5;
```

hook1 for NF_INET_PRE_ROUTING
hook2 for NF_INET_LOCAL_IN
hook3 for NF_INET_FORWARD
hook4 for NF_INET_LOCAL_OUT
hook5 for NF_INET_POST_ROUTING

Now, create the *register* filter hooks for the above mentioned hooks under *registerFilter* function

```
//NF_INET_PRE_ROUTING
   hook1.hook = printInfo;
   hook1.hooknum = NF_INET_PRE_ROUTING;
   hook1.pf = PF_INET;
   hook1.priority = NF_IP_PRI_FIRST;
   nf_register_net_hook(&init_net, &hook1);
//NF_INET_LOCAL_IN
   hook2.hook = printInfo;
   hook2.hooknum = NF_INET_LOCAL_IN;
   hook2.pf = PF_INET;
   hook2.priority = NF_IP_PRI_FIRST;
   nf_register_net_hook(&init_net, &hook2);
//NF_INET_FORWARD
   hook3.hook = printInfo;
   hook3.hooknum = NF_INET_FORWARD;
   hook3.pf = PF_INET;
   hook3.priority = NF_IP_PRI_FIRST;
   nf_register_net_hook(&init_net, &hook3);
//NF_INET_LOCAL_OUT
   hook4.hook = printInfo;
   hook4.hooknum = NF_INET_LOCAL_OUT;
   hook4.pf = PF_INET;
   hook4.priority = NF_IP_PRI_FIRST;
   nf_register_net_hook(&init_net, &hook4);
//NF_INET_POST_ROUTING
   hook5.hook = printInfo;
   hook5.hooknum = NF_INET_POST_ROUTING;
   hook5.pf = PF_INET;
   hook5.priority = NF_IP_PRI_FIRST;
   nf_register_net_hook(&init_net, &hook5);
```

Create *unregister* filters under *removeFilter* function

```
void removeFilter(void) {
   printk(KERN_INFO "The filters are being removed.\n");
   nf_unregister_net_hook(&init_net, &hook1);
   nf_unregister_net_hook(&init_net, &hook2);
   nf_unregister_net_hook(&init_net, &hook3);
   nf_unregister_net_hook(&init_net, &hook4);
   nf_unregister_net_hook(&init_net, &hook5);
}
```

Compile *seedFilter_1.c* file into loadable kernel module *seedFilter_1.ko*

```
[03/29/23]seed@VM:~/.../packet_filter$ make
make -C /lib/modules/5.4.0-54-generic/build M=/home/seed/Downloads/Labsetup/Files/packet_filter modules
make[1]: Entering directory '/usr/src/linux-headers-5.4.0-54-generic'
  CC [M]  /home/seed/Downloads/Labsetup/Files/packet_filter/seedFilter_1.o
  Building modules, stage 2.
  MODPOST 1 modules
  CC [M]  /home/seed/Downloads/Labsetup/Files/packet_filter/seedFilter_1.mod.o
  LD [M]  /home/seed/Downloads/Labsetup/Files/packet_filter/seedFilter_1.ko
make[1]: Leaving directory '/usr/src/linux-headers-5.4.0-54-generic'
[03/29/23]seed@VM:~/.../packet_filter$ ls -lrt
total 44
-rw-rw-r-- 1 seed seed 2770 Mar 29 21:22 seedFilter.c
-rw-rw-r-- 1 seed seed 2770 Mar 29 21:39 seedFilter_1.c
-rw-rw-r-- 1 seed seed  261 Mar 29 21:41 Makefile
-rw-rw-r-- 1 seed seed 5160 Mar 29 21:57 seedFilter_1.o
-rw-rw-r-- 1 seed seed   66 Mar 29 21:57 seedFilter_1.mod
-rw-rw-r-- 1 seed seed   66 Mar 29 21:57 modules.order
-rw-rw-r-- 1 seed seed  560 Mar 29 21:57 seedFilter_1.mod.c
-rw-rw-r-- 1 seed seed    0 Mar 29 21:57 Module.symvers
-rw-rw-r-- 1 seed seed 2808 Mar 29 21:57 seedFilter_1.mod.o
-rw-rw-r-- 1 seed seed 7096 Mar 29 21:57 seedFilter_1.ko
```

Before registering the new filters to the kernel, let's remove the old *seedFilter.ko* filters registered to avoid discrepancies

```
[03/29/23]seed@VM:~/.../packet_filter$ lsmod | grep seedFilter
seedFilter             16384  0
[03/29/23]seed@VM:~/.../packet_filter$ sudo rmmod seedFilter.ko
[03/29/23]seed@VM:~/.../packet_filter$ lsmod | grep seedFilter
```

Check the messages - dmesg

```
[ 7569.440354]      10.0.2.15  --> 224.0.0.251 (UDP)
[ 7626.628186] The filters are being removed.
```

Load the module (inserting module), List module - *seedFilter_1*

```
[03/29/23]seed@VM:~/.../packet_filter$ sudo insmod seedFilter_1.ko
[03/29/23]seed@VM:~/.../packet_filter$ lsmod | grep seedFilter_1
seedFilter_1           16384  0
```

Successfully loaded *seedFilter_1* Loadable kernel module into the kernel.

Check the messages - dmesg

```
[ 8629.728075] Registering filters.
[ 8768.457168] *** LOCAL_OUT
[ 8768.457175]     10.0.2.15  --> 8.8.4.4 (UDP)
[ 8768.457205] *** POST_ROUTING
[ 8768.457208]     10.0.2.15  --> 8.8.4.4 (UDP)
[ 8768.566109] *** PRE_ROUTING
[ 8768.566134]     8.8.4.4  --> 10.0.2.15 (UDP)
[ 8768.566148] *** LOCAL_IN
[ 8768.566151]     8.8.4.4  --> 10.0.2.15 (UDP)
```

Filters started registering.

From the experiment results i understand that

- *NF_INET_PRE_ROUTING* is called before an IP packet is routed from computer/machine to server
- *NF_INET_LOCAL_IN* is called on an incoming IP packet
- *NF_INET_FORWARD* is called after routing, it is found that the packet is for another networking domain, and not for a local process (i didn't observed in the docker messages)
- *NF_INET_LOCAL_OUT* is called on an outgoing IP packet
- *NF_INET*_POST_ROUTING is called before an outgoing IP packet leaves the computer

Remove the *seedFilter_1* mod to avoid discrepancies.

```
[03/29/23]seed@VM:~/.../packet_filter$ sudo rmmod seedFilter_1.ko
[03/29/23]seed@VM:~/.../packet_filter$ lsmod | grep seedFilter_1
[03/29/23]seed@VM:~/.../packet_filter$
```

Check the messages - dmesg

```
[10365.602670] *** LOCAL_IN
[10365.602671]     35.164.39.216  --> 10.0.2.15 (TCP)
[10380.803740] The filters are being removed.
```

Task 3.
Remove all the make files

```
[03/29/23]seed@VM:~/.../packet_filter$ make clean
make -C /lib/modules/5.4.0-54-generic/build M=/home/seed/Downloads/Labsetup/Files/packet_filter clean
make[1]: Entering directory '/usr/src/linux-headers-5.4.0-54-generic'
  CLEAN   /home/seed/Downloads/Labsetup/Files/packet_filter/Module.symvers
make[1]: Leaving directory '/usr/src/linux-headers-5.4.0-54-generic'
[03/29/23]seed@VM:~/.../packet_filter$ ls -lrt
total 12
-rw-rw-r-- 1 seed seed 2770 Mar 29 21:22 seedFilter.c
-rw-rw-r-- 1 seed seed  261 Mar 29 22:20 Makefile
-rw-rw-r-- 1 seed seed 3540 Mar 29 22:33 seedFilter_1.c
```

In task 3, implement two more hooks to preventing other computers to ping the VM, and preventing other computers to telnet into the VM

Create a copy of the *seedfilter.c*, so the original file will not get disturbed by the changes made.

```
[03/29/23]seed@VM:~/.../packet_filter$ cp seedFilter.c seedFilter_2.c
[03/29/23]seed@VM:~/.../packet_filter$ ls -lrt
total 16
-rw-rw-r-- 1 seed seed 2770 Mar 29 21:22 seedFilter.c
-rw-rw-r-- 1 seed seed  261 Mar 29 22:20 Makefile
-rw-rw-r-- 1 seed seed 3540 Mar 29 22:33 seedFilter_1.c
-rw-rw-r-- 1 seed seed 2770 Mar 29 23:02 seedFilter_2.c
[03/29/23]seed@VM:~/.../packet_filter$
```

Change the file name in *Makefile*

```
#obj-m += seedFilter.o
#obj-m += seedFilter_1.o
obj-m += seedFilter_2.o

all:
	make -C /lib/modules/$(shell uname -r)/build M=$(PWD) modules

clean:
	make -C /lib/modules/$(shell uname -r)/build M=$(PWD) clean

ins:
	sudo dmesg -C
	sudo insmod seedFilter.ko

rm:
	sudo rmmod seedFilter
```

Implementing new 2 more hooks

```
static struct nf_hook_ops hook1, hook2, hook3, hook4;
```

hook1 for *printInfo()* function
hook2 for *blockUDP()* function
hook3 for *blockICMP()* function
hook4 for *blockTelnet()* function

Create a function *blockICMP*, which is blocking ping to VM

In this function,

- We are blocking ICMP traffic with a type of ICMP_ECHO (which is commonly used for ping requests) and sent to the IP address "10.9.0.1" which is of attacker machine
- It converts the IP address from dotted decimal notation to 32-bit binary and compares it with the destination IP address in the packet.
- If both the destination IP address and the ICMP type match the predefined values, the packet is dropped, and a warning message is printed to the kernel log.

```
//blocking ping to VM
unsigned int blockICMP(void *priv, struct sk_buff *skb,
                       const struct nf_hook_state *state)
{
   struct iphdr *iph;
   struct icmphdr *icmph;

   //u16  port    = 53;
   char ip[16] = "10.9.0.1";
   u32  ip_addr;

   if (!skb) return NF_ACCEPT;

   iph = ip_hdr(skb);
   // Convert the IPv4 address from dotted decimal to 32-bit binary
   in4_pton(ip, -1, (u8 *)&ip_addr, '\0', NULL);

   if (iph->protocol == IPPROTO_ICMP) {
       icmph = icmp_hdr(skb);
       if (iph->daddr == ip_addr && icmph->type == ICMP_ECHO){
            printk(KERN_WARNING "*** Dropping %pI4 (ICMP)\n", &(iph->daddr));
            return NF_DROP;
        }
   }
   return NF_ACCEPT;
}
```

Create a function *blockTelnet*, which is blocking ping to VM

In this function,

- The IP address and port number to block are defined, along with variables for the IP header and TCP header of the incoming packet.
- The function then checks whether the packet is a TCP packet and whether its destination IP address and port match the predefined values.
- If there is a match, the function drops the packet and logs a warning message to the kernel log. Otherwise, it accepts the packet and allows it to pass through.

```c
//blocking telnet to VM
unsigned int blockTelnet(void *priv, struct sk_buff *skb,
                         const struct nf_hook_state *state)
{
   struct iphdr *iph;
   struct tcphdr *tcph;

   u16  port   = 23;
   char ip[16] = "10.9.0.1";
   u32  ip_addr;

   if (!skb) return NF_ACCEPT;

   iph = ip_hdr(skb);
   // Convert the IPv4 address from dotted decimal to 32-bit binary
   in4_pton(ip, -1, (u8 *)&ip_addr, '\0', NULL);

   if (iph->protocol == IPPROTO_TCP) {
       tcph = tcp_hdr(skb);
       if (iph->daddr == ip_addr && ntohs(tcph->dest) == port){
            printk(KERN_WARNING "*** Dropping %pI4 (TCP), port %d\n", &(iph->daddr), port);
            return NF_DROP;
        }
   }
   return NF_ACCEPT;
}
```

Created additional *registerFilter* hooks for *blockICMP* and *blockTelnet* functions

```c
int registerFilter(void) {
    printk(KERN_INFO "Registering filters.\n");

    hook1.hook = printInfo;
    hook1.hooknum = NF_INET_LOCAL_OUT;
    hook1.pf = PF_INET;
    hook1.priority = NF_IP_PRI_FIRST;
    nf_register_net_hook(&init_net, &hook1);

    hook2.hook = blockUDP;
    hook2.hooknum = NF_INET_POST_ROUTING;
    hook2.pf = PF_INET;
    hook2.priority = NF_IP_PRI_FIRST;
    nf_register_net_hook(&init_net, &hook2);

    hook3.hook = blockICMP;
    hook3.hooknum = NF_INET_PRE_ROUTING;
    hook3.pf = PF_INET;
    hook3.priority = NF_IP_PRI_FIRST;
    nf_register_net_hook(&init_net, &hook3);

    hook4.hook = blockTelnet;
    hook4.hooknum = NF_INET_PRE_ROUTING;
    hook4.pf = PF_INET;
    hook4.priority = NF_IP_PRI_FIRST;
    nf_register_net_hook(&init_net, &hook4);

    return 0;
}
```

Created additional *removeFilter* hooks for *blockICMP* and *blockTelnet* functions to remove the filters

```
void removeFilter(void) {
    printk(KERN_INFO "The filters are being removed.\n");
    nf_unregister_net_hook(&init_net, &hook1);
    nf_unregister_net_hook(&init_net, &hook2);
    nf_unregister_net_hook(&init_net, &hook3);
    nf_unregister_net_hook(&init_net, &hook4);
}
```

Compile *seedFilter_2.c* file into loadable kernel module *seedFilter_2.ko*

```
[03/30/23]seed@VM:~/.../packet_filter$ make
make -C /lib/modules/5.4.0-54-generic/build M=/home/seed/Downloads/Labsetup/Files/packet_filter modules
make[1]: Entering directory '/usr/src/linux-headers-5.4.0-54-generic'
  CC [M]  /home/seed/Downloads/Labsetup/Files/packet_filter/seedFilter_2.o
  Building modules, stage 2.
  MODPOST 1 modules
  CC [M]  /home/seed/Downloads/Labsetup/Files/packet_filter/seedFilter_2.mod.o
  LD [M]  /home/seed/Downloads/Labsetup/Files/packet_filter/seedFilter_2.ko
make[1]: Leaving directory '/usr/src/linux-headers-5.4.0-54-generic'
[03/30/23]seed@VM:~/.../packet_filter$ ls -lrt
total 56
-rw-rw-r-- 1 seed seed 2770 Mar 29 21:22 seedFilter.c
-rw-rw-r-- 1 seed seed 3540 Mar 29 22:33 seedFilter_1.c
-rw-rw-r-- 1 seed seed  287 Mar 29 23:10 Makefile
-rw-rw-r-- 1 seed seed 4652 Mar 30 01:48 seedFilter_2.c
-rw-rw-r-- 1 seed seed 6848 Mar 30 01:48 seedFilter_2.o
-rw-rw-r-- 1 seed seed   66 Mar 30 01:48 seedFilter_2.mod
-rw-rw-r-- 1 seed seed   66 Mar 30 01:48 modules.order
-rw-rw-r-- 1 seed seed  560 Mar 30 01:48 seedFilter_2.mod.c
-rw-rw-r-- 1 seed seed    0 Mar 30 01:48 Module.symvers
-rw-rw-r-- 1 seed seed 2808 Mar 30 01:48 seedFilter_2.mod.o
-rw-rw-r-- 1 seed seed 8744 Mar 30 01:48 seedFilter_2.ko
```

Load the module (inserting module), List module - *seedFilter_2*

```
[03/30/23]seed@VM:~/.../packet_filter$ sudo insmod seedFilter_2.ko
[03/30/23]seed@VM:~/.../packet_filter$ lsmod | grep seedFilter_2
seedFilter_2           16384  0
[03/30/23]seed@VM:~/.../packet_filter$
```

Check the messages - dmesg

```
[19294.157385] Registering filters.
[19374.092917] *** LOCAL_OUT
[19374.092920]     10.0.2.15   --> 35.163.182.76 (TCP)
[19379.203136] *** LOCAL_OUT
[19379.203142]     127.0.0.1   --> 127.0.0.53 (UDP)
[19379.204005] *** LOCAL_OUT
[19379.204009]     10.0.2.15   --> 75.75.75.75 (UDP)
[19379.204198] *** LOCAL_OUT
[19379.204202]     127.0.0.1   --> 127.0.0.53 (UDP)
[19379.207341] *** LOCAL_OUT
[19379.207348]     10.0.2.15   --> 75.75.75.75 (UDP)
```

Testing the filters

Connecting to host-A shell and testing the connection with 10.9.0.1 before the loading the module into kernel

Ping

```
[03/30/23]seed@VM:~/.../packet_filter$ docksh hostA-10.9.0.5
root@56f473f43aa8:/# ping 10.9.0.1
PING 10.9.0.1 (10.9.0.1) 56(84) bytes of data.
64 bytes from 10.9.0.1: icmp_seq=1 ttl=64 time=0.063 ms
64 bytes from 10.9.0.1: icmp_seq=2 ttl=64 time=0.165 ms
64 bytes from 10.9.0.1: icmp_seq=3 ttl=64 time=0.156 ms
64 bytes from 10.9.0.1: icmp_seq=4 ttl=64 time=0.123 ms
^C
--- 10.9.0.1 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3071ms
rtt min/avg/max/mdev = 0.063/0.126/0.165/0.040 ms
root@56f473f43aa8:/#
```

Before loading the module into the kernel, the ping working fine

Telnet

```
root@56f473f43aa8:/# telnet 10.9.0.1
Trying 10.9.0.1...
Connected to 10.9.0.1.
Escape character is '^]'.
Ubuntu 20.04.1 LTS
VM login: seed
Password:
Welcome to Ubuntu 20.04.1 LTS (GNU/Linux 5.4.0-54-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage

0 updates can be installed immediately.
0 of these updates are security updates.


The list of available updates is more than a week old.
To check for new updates run: sudo apt update
Your Hardware Enablement Stack (HWE) is supported until April 2025.
Last login: Thu Mar 30 02:15:11 EDT 2023 from www.SeedLabSQLInjection.com on pts/4
```

Before loading the module into the kernel, the telnet connection working fine

After loading the module into the kernel

Ping

```
[03/30/23]seed@VM:~/.../packet_filter$ docksh hostA-10.9.0.5
root@56f473f43aa8:/# ping 10.9.0.1
PING 10.9.0.1 (10.9.0.1) 56(84) bytes of data.
^C
--- 10.9.0.1 ping statistics ---
6 packets transmitted, 0 received, 100% packet loss, time 5103ms

root@56f473f43aa8:/#
```

Total of 6 packets transmitted to the attacker machine (10.9.0.1), due to the filters defined there is 100% loss in the packet transmission.

Check the messages - dmesg

```
[21427.100046] Registering filters.
[21440.305513] *** Dropping 10.9.0.1 (ICMP)
[21441.320404] *** Dropping 10.9.0.1 (ICMP)
[21442.344287] *** Dropping 10.9.0.1 (ICMP)
[21443.370806] *** Dropping 10.9.0.1 (ICMP)
[21444.392333] *** Dropping 10.9.0.1 (ICMP)
[21445.416893] *** Dropping 10.9.0.1 (ICMP)
```

We can see that each packet transmitted is getting dropped because of the filter implemented.

Telnet

```
root@56f473f43aa8:/# telnet 10.9.0.1
Trying 10.9.0.1...
^C
root@56f473f43aa8:/#
```

Telnet connection is kept on waiting till it is manually interrupted.

Check the messages - dmesg

```
[21496.227248] *** LOCAL_OUT
[21496.227249]     127.0.0.53  --> 127.0.0.1 (UDP)
[21508.299310] *** Dropping 10.9.0.1 (TCP), port 23
[21509.321025] *** Dropping 10.9.0.1 (TCP), port 23
[21511.337847] *** Dropping 10.9.0.1 (TCP), port 23
```

When telnet is trying to connect with attacker machine (10.9.0.1), every TCP packet is getting dropped.

Removing filters to avoid the discrepancies

```
[03/30/23]seed@VM:~/.../packet_filter$ sudo rmmod seedFilter_2
[03/30/23]seed@VM:~/.../packet_filter$ lsmod | grep seedFilter_2
[03/30/23]seed@VM:~/.../packet_filter$
```

```
[21519.331337] *** LOCAL_OUT
[21519.331341]     10.0.2.15  --> 185.125.190.18 (TCP)
[21663.810209] The filters are being removed.
```

Task 2: Experimenting with Stateless Firewall Rules

Task 2.A: Protecting the Router

Login to the *seed-router*

```
[03/30/23]seed@VM:~/.../packet_filter$ docksh seed-router
root@f863f0cb47dd:/#
```

There are 2 interfaces to the router with IP addresses - 10.9.0.11 & 192.168.60.11

Test the connection with router by *ping*

Interface - 1

```
root@56f473f43aa8:/# [03/30/23]seed@VM:~/.../packet_filter$ ping 10.9.0.11
PING 10.9.0.11 (10.9.0.11) 56(84) bytes of data.
64 bytes from 10.9.0.11: icmp_seq=1 ttl=64 time=0.060 ms
64 bytes from 10.9.0.11: icmp_seq=2 ttl=64 time=0.117 ms
64 bytes from 10.9.0.11: icmp_seq=3 ttl=64 time=0.131 ms
64 bytes from 10.9.0.11: icmp_seq=4 ttl=64 time=0.041 ms
64 bytes from 10.9.0.11: icmp_seq=5 ttl=64 time=0.039 ms
^C
--- 10.9.0.11 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4090ms
rtt min/avg/max/mdev = 0.039/0.077/0.131/0.038 ms
```

Interface - 2

```
root@56f473f43aa8:/# ping 192.168.60.11
PING 192.168.60.11 (192.168.60.11) 56(84) bytes of data.
64 bytes from 192.168.60.11: icmp_seq=1 ttl=64 time=0.109 ms
64 bytes from 192.168.60.11: icmp_seq=2 ttl=64 time=0.186 ms
64 bytes from 192.168.60.11: icmp_seq=3 ttl=64 time=0.186 ms
64 bytes from 192.168.60.11: icmp_seq=4 ttl=64 time=0.186 ms
64 bytes from 192.168.60.11: icmp_seq=5 ttl=64 time=0.181 ms
^C
--- 192.168.60.11 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4073ms
rtt min/avg/max/mdev = 0.109/0.169/0.186/0.030 ms
```

Checking the default rules of iptables in the router

```
root@f863f0cb47dd:/# iptables -t filter -L -n
Chain INPUT (policy DROP)
target     prot opt source               destination

Chain FORWARD (policy ACCEPT)
target     prot opt source               destination

Chain OUTPUT (policy ACCEPT)
target     prot opt source               destination
```

It is good that the ip filter table is empty before performing the tasks.

Accepting input icmp packets

```
root@f863f0cb47dd:/# iptables -A INPUT  -p icmp --icmp-type echo-request -j ACCEPT
root@f863f0cb47dd:/# iptables -t filter -L -n
Chain INPUT (policy DROP)
target     prot opt source               destination
ACCEPT     icmp --  0.0.0.0/0            0.0.0.0/0            icmptype 8

Chain FORWARD (policy ACCEPT)
target     prot opt source               destination

Chain OUTPUT (policy ACCEPT)
target     prot opt source               destination
root@f863f0cb47dd:/#
```

Here, I implemented a rule to allow incoming "ping" requests to the system by adding a rule to the INPUT chain of the firewall, which is responsible for filtering incoming packets.

- The rule matches any incoming ICMP Echo Request packets, which have an ICMP type value of 8, using the --icmp-type option.
- The -p icmp option specifies that the rule applies to ICMP traffic.
- The -j ACCEPT option tells the firewall to accept any packets that match the rule.

Accepting output icmp packets

```
root@f863f0cb47dd:/# iptables -A OUTPUT -p icmp --icmp-type echo-reply -j ACCEPT
root@f863f0cb47dd:/# iptables -t filter -L -n
Chain INPUT (policy DROP)
target     prot opt source               destination
ACCEPT     icmp --  0.0.0.0/0            0.0.0.0/0            icmptype 8

Chain FORWARD (policy ACCEPT)
target     prot opt source               destination

Chain OUTPUT (policy ACCEPT)
target     prot opt source               destination
ACCEPT     icmp --  0.0.0.0/0            0.0.0.0/0            icmptype 0
```

Here, I implemented a rule to allow outgoing "ping" requests to the system by adding a rule to the OUTPUT chain of the firewall, which is responsible for filtering outgoing packets.

- The rule matches any outgoing ICMP Echo Reply packets ,which have an ICMP type value of 0, using the --icmp-type option.

- The -p icmp option specifies that the rule applies to ICMP traffic.
- The -j ACCEPT option tells the firewall to accept any packets that match the rule.

default rule for OUTPUT

```
root@f863f0cb47dd:/# iptables -P OUTPUT DROP
root@f863f0cb47dd:/# iptables -t filter -L -n
Chain INPUT (policy DROP)
target     prot opt source               destination
ACCEPT     icmp --  0.0.0.0/0            0.0.0.0/0            icmptype 8

Chain FORWARD (policy ACCEPT)
target     prot opt source               destination

Chain OUTPUT (policy DROP)
target     prot opt source               destination
ACCEPT     icmp --  0.0.0.0/0            0.0.0.0/0            icmptype 0
```

Here, I implemented a rule to set the default policy for the OUTPUT chain of the firewall to DROP

- Any outgoing traffic that does not match an explicit rule allowing it will be dropped by default

default rule for INPUT

```
root@f863f0cb47dd:/# iptables -P INPUT DROP
root@f863f0cb47dd:/# iptables -t filter -L -n
Chain INPUT (policy DROP)
target     prot opt source               destination
ACCEPT     icmp --  0.0.0.0/0            0.0.0.0/0            icmptype 8

Chain FORWARD (policy ACCEPT)
target     prot opt source               destination

Chain OUTPUT (policy DROP)
target     prot opt source               destination
ACCEPT     icmp --  0.0.0.0/0            0.0.0.0/0            icmptype 0
```

Here, I implemented a rule to set the default policy for the INPUT chain of the firewall to DROP

- Any incoming traffic that does not match an explicit rule allowing it will be dropped by default

Testing the connections

Ping

```
root@56f473f43aa8:/# ping 10.9.0.11
PING 10.9.0.11 (10.9.0.11) 56(84) bytes of data.
64 bytes from 10.9.0.11: icmp_seq=1 ttl=64 time=0.069 ms
64 bytes from 10.9.0.11: icmp_seq=2 ttl=64 time=0.153 ms
64 bytes from 10.9.0.11: icmp_seq=3 ttl=64 time=0.196 ms
64 bytes from 10.9.0.11: icmp_seq=4 ttl=64 time=0.079 ms
^C
--- 10.9.0.11 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3073ms
rtt min/avg/max/mdev = 0.069/0.124/0.196/0.052 ms
root@56f473f43aa8:/#
```

After setting the rules, the *ping* is working as expected with 0% packet loss.

Telnet

```
root@56f473f43aa8:/# telnet 10.9.0.11
Trying 10.9.0.11...
^C
root@56f473f43aa8:/#
```

As per the rules, except *ping* utility no other utility should work i.e., *telnet*
As expected *telnet* is not able to connect to the router interface.

Restore the filter table to its original state

```
root@f863f0cb47dd:/# iptables -F
root@f863f0cb47dd:/# iptables -P INPUT ACCEPT
root@f863f0cb47dd:/# iptables -P OUTPUT ACCEPT
root@f863f0cb47dd:/# iptables -t filter -L -n
Chain INPUT (policy ACCEPT)
target     prot opt source               destination

Chain FORWARD (policy ACCEPT)
target     prot opt source               destination

Chain OUTPUT (policy ACCEPT)
target     prot opt source               destination
```

It is advised to restore the filter table to its original state, so that there will be no discrepancies while performing other tasks

Observations

- The ICMP packets can transmit successfully without any loss because when passing the rules to the iptables, I mentioned to ACCEPT ICMP packets both incoming and outgoing through the firewall
- The TCP packets which are for telnet will drop because when passing the rules to iptables, I mentioned to DROP every other packet except ICMP.

Task 2.B: Protecting the Internal Network

1. Outside hosts cannot ping internal hosts

Dropping/Denying ICMP packets

```
root@f863f0cb47dd:/# iptables -A FORWARD -i eth0 -p icmp --icmp-type echo-request -j DROP
root@f863f0cb47dd:/# iptables -t filter -L -n
Chain INPUT (policy ACCEPT)
target     prot opt source               destination

Chain FORWARD (policy ACCEPT)
target     prot opt source               destination
DROP       icmp --  0.0.0.0/0            0.0.0.0/0            icmptype 8

Chain OUTPUT (policy ACCEPT)
target     prot opt source               destination
root@f863f0cb47dd:/#
```

Here i implemented a rule, that adds a new rule to the FORWARD chain of the firewall, which is responsible for filtering packets that are being forwarded between network interfaces

- This rule specifies that any incoming ICMP Echo Request packets received on the network interface should be dropped.

To test the connection, logging as an internal host

```
[03/30/23]seed@VM:~/.../packet_filter$ docksh host1-192.168.60.5
root@06d66a6c664d:/#
```

Connected to internal host to test the rule

From outside host, pinging the internal host

```
root@56f473f43aa8:/# ping 192.168.60.5
PING 192.168.60.5 (192.168.60.5) 56(84) bytes of data.
^C
--- 192.168.60.5 ping statistics ---
4 packets transmitted, 0 received, 100% packet loss, time 3058ms

root@56f473f43aa8:/#
```

As expected, there is a 100% packet loss, while pinging the internal host (192.168.60.5) from outside host (10.9.0.5)

2. Outside hosts can ping the router.

From outside host, pinging the router

```
[03/30/23]seed@VM:~/.../packet_filter$ docksh hostA-10.9.0.5
root@56f473f43aa8:/# ping 10.9.0.11
PING 10.9.0.11 (10.9.0.11) 56(84) bytes of data.
64 bytes from 10.9.0.11: icmp_seq=1 ttl=64 time=0.082 ms
64 bytes from 10.9.0.11: icmp_seq=2 ttl=64 time=0.175 ms
64 bytes from 10.9.0.11: icmp_seq=3 ttl=64 time=0.050 ms
64 bytes from 10.9.0.11: icmp_seq=4 ttl=64 time=0.185 ms
^C
--- 10.9.0.11 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3061ms
rtt min/avg/max/mdev = 0.050/0.123/0.185/0.058 ms
root@56f473f43aa8:/#
```

As expected, there is 0% loss while pinging the router(10.9.0.11) from outside host (hostA-10.9.0.5)

3. Internal hosts can ping outside hosts.

From internal host, pinging the outside host

```
[03/30/23]seed@VM:~/.../packet_filter$ docksh host1-192.168.60.5
root@06d66a6c664d:/# ping 10.9.0.5
PING 10.9.0.5 (10.9.0.5) 56(84) bytes of data.
64 bytes from 10.9.0.5: icmp_seq=1 ttl=63 time=0.179 ms
64 bytes from 10.9.0.5: icmp_seq=2 ttl=63 time=0.234 ms
64 bytes from 10.9.0.5: icmp_seq=3 ttl=63 time=0.234 ms
64 bytes from 10.9.0.5: icmp_seq=4 ttl=63 time=0.243 ms
^C
--- 10.9.0.5 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3048ms
rtt min/avg/max/mdev = 0.179/0.222/0.243/0.025 ms
root@06d66a6c664d:/#
```

As expected there is 0% packet loss while pinging the outside host (10.9.0.5) from internal host (192.168.60.5)

4. All other packets between the internal and external networks should be blocked.

```
root@f863f0cb47dd:/# iptables -A FORWARD -i eth1 -p icmp --icmp-type echo-request -j ACCEPT
root@f863f0cb47dd:/# iptables -A FORWARD -i eth0 -p icmp --icmp-type echo-reply -j ACCEPT
root@f863f0cb47dd:/# iptables -P FORWARD DROP
root@f863f0cb47dd:/# iptables -t filter -L -n
Chain INPUT (policy ACCEPT)
target     prot opt source               destination

Chain FORWARD (policy DROP)
target     prot opt source               destination
DROP       icmp --  0.0.0.0/0            0.0.0.0/0            icmptype 8
ACCEPT     icmp --  0.0.0.0/0            0.0.0.0/0            icmptype 8
ACCEPT     icmp --  0.0.0.0/0            0.0.0.0/0            icmptype 0

Chain OUTPUT (policy ACCEPT)
target     prot opt source               destination
```

Here i implemented 3 rules to block all other packets except ICMP gets blocked

- In the first rule I added a new rule to the FORWARD chain of the firewall, which allows a ping request to be initiated from a host on the eth1 network and sent to a host on the eth0 network. It applies to packets that are being forwarded through the system from eth1 to eth0.
- In the second rule I added a new rule to the FORWARD chain of the firewall, which allows incoming ICMP Echo Reply packets on the eth0 interface, which means that it allows the response to a ping request that was initiated from a host on the eth1 network to be forwarded back to the host on eth1. It applies to packets that are being forwarded through the system from eth0 to eth1.
- In the third rule I added a new rule which sets the default policy for the FORWARD chain to DROP that is not explicitly allowed by rules in the FORWARD chain.

From outside host trying to connect to internal host

```
root@56f473f43aa8:/# telnet 192.168.60.5
Trying 192.168.60.5...
^C
root@56f473f43aa8:/#
```

As expected, except ICMP packets the connections are not accepting any other packets. Here we are sending TCP packets by trying to establish a connection with telnet to the internal host (192.168.60.5) from outside host (10.9.0.5)

From internal host trying to connect to outside host

```
root@06d66a6c664d:/# telnet 10.9.0.5
Trying 10.9.0.5...
^C
root@06d66a6c664d:/#
```

As expected, except ICMP packets the connections are not accepting any other packets.

Here we are sending TCP packets by trying to establish a connection with telnet to the outside host (10.9.0.5) from the internal host (192.168.60.5).

Restore the filter table to its original state

```
root@f863f0cb47dd:/# iptables -F
root@f863f0cb47dd:/# iptables -P FORWARD ACCEPT
root@f863f0cb47dd:/# iptables -t filter -L -n
Chain INPUT (policy ACCEPT)
target     prot opt source               destination

Chain FORWARD (policy ACCEPT)
target     prot opt source               destination

Chain OUTPUT (policy ACCEPT)
target     prot opt source               destination
```

Task 2.C: Protecting Internal Servers

1. All the internal hosts run a telnet server (listening to port 23). Outside hosts can only access the telnet server on 192.168.60.5, not the other internal hosts.

To achieve this, we need to setup a rule
These rules will allow us to connect to 192.168.60.5 only

```
root@f863f0cb47dd:/# iptables -A FORWARD -i eth0 -d 192.168.60.5 -p tcp --dport 23 -j ACCEPT
root@f863f0cb47dd:/# iptables -A FORWARD -i eth1 -s 192.168.60.5 -p tcp --sport 23 -j ACCEPT
root@f863f0cb47dd:/# iptables -P FORWARD DROP
root@f863f0cb47dd:/# iptables -t filter -L -n
Chain INPUT (policy ACCEPT)
target     prot opt source               destination

Chain FORWARD (policy DROP)
target     prot opt source               destination
ACCEPT     tcp  --  0.0.0.0/0            192.168.60.5         tcp dpt:23
ACCEPT     tcp  --  192.168.60.5         0.0.0.0/0            tcp spt:23

Chain OUTPUT (policy ACCEPT)
target     prot opt source               destination
```

Here I implemented 3 rules, which will explicitly connects to internal host 192.168.60.5 only

- The first rule adds a rule to the FORWARD chain of the iptables firewall. The rule specifies that any TCP traffic that is destined for port 23 on the IP address 192.168.60.5 and enters the network interface eth0 should be accepted
- The second rule adds a rule to the FORWARD chain of the iptables firewall. The rule allows incoming TCP traffic from source port 23 (default Telnet port) on interface eth0 and with source IP address 192.168.60.5
- The third rule adds a default rule to the FORWARD chain of the iptables firewall, if no rule in the FORWARD chain matches a packet, the packet will be dropped by default

Connecting to internal host 192.168.60.5 via telnet from outside host 10.9.0.5

```
root@56f473f43aa8:/# telnet 192.168.60.5
Trying 192.168.60.5...
Connected to 192.168.60.5.
Escape character is '^]'.
Ubuntu 20.04.1 LTS
06d66a6c664d login: seed
Password:
Welcome to Ubuntu 20.04.1 LTS (GNU/Linux 5.4.0-54-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage

This system has been minimized by removing packages and content that are
not required on a system that users do not log into.

To restore this content, you can run the 'unminimize' command.

The programs included with the Ubuntu system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Ubuntu comes with ABSOLUTELY NO WARRANTY, to the extent permitted by
applicable law.

seed@06d66a6c664d:~$
```

As expected, from outside host (10.9.0.5) we can connect to the internal host (192.168.60.5) with telnet

Connecting to internal host 192.168.60.6 and 192.168.60.7 via telnet from outside host 10.9.0.5

```
root@56f473f43aa8:/# telnet 192.168.60.6
Trying 192.168.60.6...
^C
root@56f473f43aa8:/# telnet 192.168.60.7
Trying 192.168.60.7...
^C
root@56f473f43aa8:/#
```

As expected, from outside host (10.9.0.5) we are not able to connect to the internal hosts (192.168.60.6 and 192.168.60.7) except 192.168.60.5

2. Outside hosts cannot access other internal servers.

To test this we can use NetCat command (nc)
Setting up a listener at internal host

```
root@06d66a6c664d:/# nc -lt 5050
```

Setting up the listener at the internal host end (192.168.60.5)

Trying to connect to the internal server from external host

```
root@56f473f43aa8:/# nc 192.168.60.5 5050
hello
```

Starting the NetCat service on the external host end (10.9.0.5) and passing a packet/message called ‘hello’ to the internal server on the host (192.168.60.5)

At listener side

```
root@06d66a6c664d:/# nc -lt 5050
```

As expected no packets received at internal server from outside host (10.9.0.5)

3. Internal hosts can access all the internal servers.

To test this, logging into second internal container

```
[03/30/23]seed@VM:~/.../packet_filter$ docksh host2-192.168.60.6
root@ff64c9ae7dae:/#
```

Logging into the docker container as in a shell mode of the internal host (192.168.60.6)

Trying to access the internal server from internal host

```
[03/30/23]seed@VM:~/.../packet_filter$ docksh host2-192.168.60.6
root@ff64c9ae7dae:/# nc 192.168.60.5 5050
hello
```

Starting the NetCat server with the internal host 192.168.60.5 and port 5050. Now, passing the ‘hello’ message to the second internal host

At listener side

```
root@06d66a6c664d:/# nc -lt 5050
hello
```

Starting the NetCat listener service with port 5050.
As expected, successfully received the 'hello' message from the internal server host (192.168.60.5) to the other internal host (192.168.60.6)

4. Internal hosts cannot access external servers.

Trying to access the external server from internal host

```
root@06d66a6c664d:/# nc 10.9.0.5 5050
hello
```

Starting NetCat service with 10.9.0.5 with port 5050 and passing message 'hello' to the external server host (10.9.0.5)

At listener side

```
[03/31/23]seed@VM:~/.../Labsetup$ docksh host1-192.168.60.5
root@06d66a6c664d:/# nc -lt 5050
```

Starting NetCat service listener at the external server side with port 5050
As expected no message received to the external host (10.9.0.5) from the internal host (192.168.60.5)

Task 3: Connection Tracking and Stateful Firewall

Task 3.A: Experiment with the Connection Tracking

ICMP Experiment

Initially there are 0 flow entries

```
root@f863f0cb47dd:/# conntrack -L
conntrack v1.4.5 (conntrack-tools): 0 flow entries have been shown.
root@f863f0cb47dd:/#
```

Executing *conntrack -L* command will

- Displays the current connection tracking entries in the Linux kernel's connection tracking table.

- This table keeps track of all currently active network connections, including TCP and UDP connections, and allows the kernel to enforce firewall rules based on the connection state.
- The output of the command includes information about the protocol, source and destination IP addresses and ports, connection state, and other details about each connection.

Pinging *ICMP* packets from 10.9.0.5 to 195.168.60.5

```
root@56f473f43aa8:/# ping 192.168.60.5
PING 192.168.60.5 (192.168.60.5) 56(84) bytes of data.
64 bytes from 192.168.60.5: icmp_seq=1 ttl=63 time=0.106 ms
64 bytes from 192.168.60.5: icmp_seq=2 ttl=63 time=0.240 ms
64 bytes from 192.168.60.5: icmp_seq=3 ttl=63 time=0.237 ms
64 bytes from 192.168.60.5: icmp_seq=4 ttl=63 time=0.099 ms
64 bytes from 192.168.60.5: icmp_seq=5 ttl=63 time=0.237 ms
64 bytes from 192.168.60.5: icmp_seq=6 ttl=63 time=0.090 ms
64 bytes from 192.168.60.5: icmp_seq=7 ttl=63 time=0.061 ms
64 bytes from 192.168.60.5: icmp_seq=8 ttl=63 time=0.235 ms
64 bytes from 192.168.60.5: icmp_seq=9 ttl=63 time=0.225 ms
64 bytes from 192.168.60.5: icmp_seq=10 ttl=63 time=0.226 ms
64 bytes from 192.168.60.5: icmp_seq=11 ttl=63 time=0.250 ms
64 bytes from 192.168.60.5: icmp_seq=12 ttl=63 time=0.299 ms
^C
--- 192.168.60.5 ping statistics ---
12 packets transmitted, 12 received, 0% packet loss, time 11262ms
rtt min/avg/max/mdev = 0.061/0.192/0.299/0.075 ms
```

As expected, 0% packet loss happened here in between external host (10.9.0.5) to the internal host (195.168.60.5)

Connection tracking information from router

```
root@f863f0cb47dd:/# conntrack -L
icmp     1 8 src=10.9.0.5 dst=192.168.60.5 type=8 code=0 id=37 src=192.168.60.5 dst=10.9.0.5 type=0 code=0 id=37 mark=0 use=1
conntrack v1.4.5 (conntrack-tools): 1 flow entries have been shown.
root@f863f0cb47dd:/# conntrack -L
conntrack v1.4.5 (conntrack-tools): 0 flow entries have been shown.
root@f863f0cb47dd:/#
```

Here, we can say that there is 1 entry in the connection track and from my analysis the connection kept for around 10 to 15 seconds

UDP Experiment

Starting netcat UDP server on 192.168.60.5

```
[03/30/23]seed@VM:~/.../Labsetup$ docksh host1-192.168.60.5
root@06d66a6c664d:/# nc -lu 9090
```

Starting NetCat UDP packet listener service with port 9090 at the external host (192.168.60.5)

Sending UDP packets from 10.9.0.5

```
[03/30/23]seed@VM:~/.../Labsetup$ docksh hostA-10.9.0.5
root@56f473f43aa8:/# nc -u  192.168.60.5 9090
hello
```

Starting remote connection using NetCat service with UDP protocol to connect to the IP address 192.168.60.5 and the port number 9090.
And sending the message ‘hello’

Receiving UDP packets on 192.168.60.5

```
[03/30/23]seed@VM:~/.../Labsetup$ docksh host1-192.168.60.5
root@06d66a6c664d:/# nc -lu 9090
hello
```

At the listener end i.e., internal host(192.168.60.5) we received the message (hello)

Connection tracking information from router

```
root@f863f0cb47dd:/# conntrack -L
udp      17 23 src=10.9.0.5 dst=192.168.60.5 sport=33209 dport=9090 [UNREPLIED] src=192.168.60.5 dst=10.9.0.5 sport=9090 dport=33209 mark=0 u
se=1
conntrack v1.4.5 (conntrack-tools): 1 flow entries have been shown.
root@f863f0cb47dd:/# conntrack -L
conntrack v1.4.5 (conntrack-tools): 0 flow entries have been shown.
root@f863f0cb47dd:/#
```

Here, we can say that there is 1 entry in the connection track and from my analysis the connection kept for around 30 seconds

TCP Experiment

Sending TCP packets from 10.9.0.5

```
[03/30/23]seed@VM:~/.../Labsetup$ docksh hostA-10.9.0.5
root@56f473f43aa8:/# nc 192.168.60.5 9090
hello
```

Starting remote connection using NetCat service with TCP protocol to connect to the IP address 192.168.60.5 and the port number 9090.
And sending the message ‘hello’

Receiving TCP packets on 192.168.60.5

```
[03/30/23]seed@VM:~/.../Labsetup$ docksh host1-192.168.60.5
root@06d66a6c664d:/# nc -l 9090
hello
```

At the listener end i.e., internal host(192.168.60.5) we received the message (hello)

Connection tracking information from router

```
root@f863f0cb47dd:/# conntrack -L
tcp      6 115 TIME_WAIT src=10.9.0.5 dst=192.168.60.5 sport=50812 dport=9090 src=192.168.60.5 dst=10.9.0.5 sport=9090 dport=50812 [ASSURED]
mark=0 use=1
conntrack v1.4.5 (conntrack-tools): 1 flow entries have been shown.
root@f863f0cb47dd:/# conntrack -L
conntrack v1.4.5 (conntrack-tools): 0 flow entries have been shown.
root@f863f0cb47dd:/#
```

Here, we can say that there is 1 entry in the connection track and from my analysis the connection kept till after the connection kills/closes manually

Task 3.B: Setting Up a Stateful Firewall

To achieve the task, we need to setup some rules on the router

```
root@f863f0cb47dd:/# iptables -A FORWARD -p tcp -i eth0 -d 192.168.60.5 --dport 23 --syn -m conntrack --ctstate NEW -j ACCEPT
root@f863f0cb47dd:/# iptables -A FORWARD -p tcp -i eth1 --syn -m conntrack --ctstate NEW -j ACCEPT
root@f863f0cb47dd:/# iptables -A FORWARD -p tcp -m conntrack --ctstate RELATED,ESTABLISHED -j ACCEPT
root@f863f0cb47dd:/# iptables -A FORWARD -p tcp -j DROP
root@f863f0cb47dd:/# iptables -P FORWARD ACCEPT
root@f863f0cb47dd:/# iptables -L -n
Chain INPUT (policy ACCEPT)
target     prot opt source               destination

Chain FORWARD (policy ACCEPT)
target     prot opt source               destination
ACCEPT     tcp  --  0.0.0.0/0            192.168.60.5         tcp dpt:23 flags:0x17/0x02 ctstate NEW
ACCEPT     tcp  --  0.0.0.0/0            0.0.0.0/0            tcp flags:0x17/0x02 ctstate NEW
ACCEPT     tcp  --  0.0.0.0/0            0.0.0.0/0            ctstate RELATED,ESTABLISHED
DROP       tcp  --  0.0.0.0/0            0.0.0.0/0

Chain OUTPUT (policy ACCEPT)
target     prot opt source               destination
root@f863f0cb47dd:/#
```

Here i implemented 5 rules

- The first rule adds a rule to the FORWARD chain of the firewall. The rule matches incoming TCP traffic on interface eth0, with destination IP address 192.168.60.5 and destination port 23. The SYN flag must be set, and the connection must be in the NEW state (i.e., the initial packet of a new connection). If the rule matches, it accepts the traffic and allows it to pass through the firewall to the destination. This rule allows incoming connections on the Telnet port (23) to the specified IP address on interface eth0.
- The second rule adds a rule to the FORWARD chain of the iptables firewall. This rule allows new incoming TCP connections on interface eth1 by checking the SYN flag in the TCP header and the connection state using the conntrack module. The --syn option matches only the TCP packets with the SYN flag set, and the --cstate NEW option matches only packets that belong to a new connection. If the packet matches both conditions, the ACCEPT target is triggered, allowing the packet to pass through the firewall.
- The third rule adds a new rule to the FORWARD chain of the iptables firewall. The rule matches all TCP packets in the connection tracking system that have a connection state of RELATED or ESTABLISHED. The RELATED state matches packets that are related to an existing connection, such as ICMP error messages or packets belonging to an FTP data channel. The ESTABLISHED state matches packets that belong to an existing connection that has already been established. The -j ACCEPT option specifies that any packets that match this rule should be accepted and allowed to continue through the firewall.

- The fourth rule adds a rule to the FORWARD chain of iptables that drops all TCP traffic being forwarded through the system. This means that any TCP packets that are trying to pass through the system will be blocked and not allowed to reach their destination
- The fourth rule sets the default policy for the FORWARD chain to ACCEPT. This means that any traffic coming through the firewall that does not match a specific rule will be allowed to pass through

Trying to connect from internal host to external server

```
[03/30/23]seed@VM:~/.../Labsetup$ docksh host1-192.168.60.5
root@06d66a6c664d:/# telnet 10.9.0.5
Trying 10.9.0.5...
Connected to 10.9.0.5.
Escape character is '^]'.
Ubuntu 20.04.1 LTS
56f473f43aa8 login: seed
Password:
Welcome to Ubuntu 20.04.1 LTS (GNU/Linux 5.4.0-54-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage

This system has been minimized by removing packages and content that are
not required on a system that users do not log into.

To restore this content, you can run the 'unminimize' command.

The programs included with the Ubuntu system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Ubuntu comes with ABSOLUTELY NO WARRANTY, to the extent permitted by
applicable law.

seed@56f473f43aa8:~$
```

As expected, after changing the rules, we can access the external host (10.9.0.5) from the internal host (192.168.60.5)

Testing all the connections by passing TCP packets from internal hosts to external host
Sending TCP packets from 192.168.60.5

```
[03/30/23]seed@VM:~/.../Labsetup$ docksh host1-192.168.60.5
root@06d66a6c664d:/# nc 10.9.0.5 9090
hello'
```

Starting remote connection using NetCat service with TCP protocol to connect to the IP address 192.168.60.5 and the port number 9090.
And sending the message ‘hello’

Receiving TCP packets on 10.9.0.5

```
[03/30/23]seed@VM:~/.../Labsetup$ docksh hostA-10.9.0.5
root@56f473f43aa8:/# nc -l 9090
hello
```

At listener end i.e., external host (10.9.0.5) the message ‘hello’ received

Sending TCP packets from 192.168.60.6

```
[03/30/23]seed@VM:~/.../Labsetup$ docksh host2-192.168.60.6
root@ff64c9ae7dae:/# nc 10.9.0.5 9090
hello
```

Starting remote connection using NetCat service with TCP protocol to connect to the IP address 192.168.60.6 and the port number 9090.
And sending the message ‘hello’

Receiving TCP packets on 10.9.0.5

```
[03/30/23]seed@VM:~/.../Labsetup$ docksh hostA-10.9.0.5
root@56f473f43aa8:/# nc -l 9090
hello
```

At listener end i.e., external host (10.9.0.5) the message ‘hello’ received

Sending TCP packets from 192.168.60.7

```
[03/30/23]seed@VM:~/.../Labsetup$ docksh host3-192.168.60.7
root@f78e15e8c54a:/# nc 10.9.0.5 9090
hello 3
```

Starting remote connection using NetCat service with TCP protocol to connect to the IP address 192.168.60.7 and the port number 9090.
And sending the message ‘hello’

Receiving TCP packets on 10.9.0.5

```
[03/30/23]seed@VM:~/.../Labsetup$ docksh hostA-10.9.0.5
root@56f473f43aa8:/# nc -l 9090
hello 3
```

At listener end i.e., external host (10.9.0.5) the message ‘hello’ received

Sending TCP packets from 10.9.0.5

```
[03/30/23]seed@VM:~/.../Labsetup$ docksh hostA-10.9.0.5
root@56f473f43aa8:/# nc 192.168.60.5 9090
hello
```

Starting a remote connection with NetCat service 192.168.60.5 with port 9090 on external host (10.9.0.5) and sending a message 'hello' to internal host (192.168.60.5)

Receiving TCP packets on 192.168.60.5

```
[03/30/23]seed@VM:~/.../Labsetup$ docksh host1-192.168.60.5
root@06d66a6c664d:/# nc -l 9090
```

As expected, no messages received at the listener end i.e., internal server (192.168.60.5) which satisfies the condition 'Outside hosts cannot access other internal servers.'

Restore the filter table to its original state

```
root@f863f0cb47dd:/# iptables -F
root@f863f0cb47dd:/# iptables -L -n
Chain INPUT (policy ACCEPT)
target     prot opt source               destination

Chain FORWARD (policy ACCEPT)
target     prot opt source               destination

Chain OUTPUT (policy ACCEPT)
target     prot opt source               destination
root@f863f0cb47dd:/#
```

Observations

- Intuitively, with a conntrack it was worked as i expected
- The disadvantage of the conntrack is that it consume more resources i.e., memory and processing power to keep track
- The advantage of the conntrack is that it allows the Linux kernel to maintain a stateful connection tracking table, which allows the firewall to keep track of the state of network connections.

Task 4: Limiting Network Traffic

Writing the first rule

```
[03/30/23]seed@VM:~/.../Labsetup$ docksh seed-router
root@f863f0cb47dd:/# iptables -A FORWARD -s 10.9.0.5 -m limit --limit 10/minute --limit-burst 5 -j ACCEPT
root@f863f0cb47dd:/# iptables -L -n
Chain INPUT (policy ACCEPT)
target     prot opt source               destination

Chain FORWARD (policy ACCEPT)
target     prot opt source               destination
ACCEPT     all  --  10.9.0.5            0.0.0.0/0            limit: avg 10/min burst 5

Chain OUTPUT (policy ACCEPT)
target     prot opt source               destination
root@f863f0cb47dd:/#
```

Here i implemented a rule to the iptables

- In the rule I added a rule to the FORWARD chain, which is responsible for packets forwarded between different interfaces.
- The rule matches packets with a source IP address of 10.9.0.5 and applies a rate limit to the traffic.
- It uses the limit module to limit the incoming traffic to a maximum of 10 packets per minute, with a limit-burst of 5 packets. This means that the first 5 packets will be accepted immediately, while any subsequent packets will be delayed until the rate limit is no longer exceeded.
- Once the rate limit is no longer exceeded, the delayed packets will be accepted.
- If the rate limit continues to be exceeded, the rule will drop the excess packets.
- Finally, the ACCEPT target is used to allow the accepted packets to pass through the firewall.

Testing the connection by ping 192.168.60.5 from 10.9.0.5

```
[03/30/23]seed@VM:~/.../Labsetup$ docksh hostA-10.9.0.5
root@56f473f43aa8:/# ping 192.168.60.5
PING 192.168.60.5 (192.168.60.5) 56(84) bytes of data.
64 bytes from 192.168.60.5: icmp_seq=1 ttl=63 time=0.109 ms
64 bytes from 192.168.60.5: icmp_seq=2 ttl=63 time=0.240 ms
64 bytes from 192.168.60.5: icmp_seq=3 ttl=63 time=0.229 ms
64 bytes from 192.168.60.5: icmp_seq=4 ttl=63 time=0.230 ms
64 bytes from 192.168.60.5: icmp_seq=5 ttl=63 time=0.232 ms
64 bytes from 192.168.60.5: icmp_seq=6 ttl=63 time=0.240 ms
64 bytes from 192.168.60.5: icmp_seq=7 ttl=63 time=0.214 ms
64 bytes from 192.168.60.5: icmp_seq=8 ttl=63 time=0.138 ms
64 bytes from 192.168.60.5: icmp_seq=9 ttl=63 time=0.247 ms
64 bytes from 192.168.60.5: icmp_seq=10 ttl=63 time=0.238 ms
64 bytes from 192.168.60.5: icmp_seq=11 ttl=63 time=0.102 ms
64 bytes from 192.168.60.5: icmp_seq=12 ttl=63 time=0.231 ms
64 bytes from 192.168.60.5: icmp_seq=13 ttl=63 time=0.240 ms
64 bytes from 192.168.60.5: icmp_seq=14 ttl=63 time=0.228 ms
64 bytes from 192.168.60.5: icmp_seq=15 ttl=63 time=0.071 ms
64 bytes from 192.168.60.5: icmp_seq=16 ttl=63 time=0.217 ms
64 bytes from 192.168.60.5: icmp_seq=17 ttl=63 time=0.117 ms
64 bytes from 192.168.60.5: icmp_seq=18 ttl=63 time=0.091 ms
64 bytes from 192.168.60.5: icmp_seq=19 ttl=63 time=0.070 ms
64 bytes from 192.168.60.5: icmp_seq=20 ttl=63 time=0.227 ms
^C
--- 192.168.60.5 ping statistics ---
20 packets transmitted, 20 received, 0% packet loss, time 19455ms
rtt min/avg/max/mdev = 0.070/0.185/0.247/0.064 ms
root@56f473f43aa8:/#
```

Here, I didn't get the response as expected. The packets are keeping on transmitting even though the limit is exceeded.

Writing the second rule

```
root@f863f0cb47dd:/# iptables -A FORWARD -s 10.9.0.5 -j DROP
root@f863f0cb47dd:/# iptables -L -n
Chain INPUT (policy ACCEPT)
target     prot opt source               destination

Chain FORWARD (policy ACCEPT)
target     prot opt source               destination
ACCEPT     all  --  10.9.0.5            0.0.0.0/0            limit: avg 10/min burst 5
DROP       all  --  10.9.0.5            0.0.0.0/0

Chain OUTPUT (policy ACCEPT)
target     prot opt source               destination
```

Here I implemented the second rule

- This rule adds a rule to the FORWARD chain of the iptables firewall.
- It specifies that any packet with a source IP address of 10.9.0.5 should be dropped. This means that any traffic originating from the IP address 10.9.0.5 will not be allowed to pass through the firewall to any destination.

Testing the connection by ping 192.168.60.5 from 10.9.0.5

```
root@56f473f43aa8:/# ping 192.168.60.5
PING 192.168.60.5 (192.168.60.5) 56(84) bytes of data.
64 bytes from 192.168.60.5: icmp_seq=1 ttl=63 time=0.108 ms
64 bytes from 192.168.60.5: icmp_seq=2 ttl=63 time=0.228 ms
64 bytes from 192.168.60.5: icmp_seq=3 ttl=63 time=0.256 ms
64 bytes from 192.168.60.5: icmp_seq=4 ttl=63 time=0.107 ms
64 bytes from 192.168.60.5: icmp_seq=5 ttl=63 time=0.227 ms
64 bytes from 192.168.60.5: icmp_seq=11 ttl=63 time=0.074 ms
64 bytes from 192.168.60.5: icmp_seq=17 ttl=63 time=0.209 ms
64 bytes from 192.168.60.5: icmp_seq=23 ttl=63 time=0.223 ms
64 bytes from 192.168.60.5: icmp_seq=29 ttl=63 time=0.249 ms
64 bytes from 192.168.60.5: icmp_seq=35 ttl=63 time=0.098 ms
64 bytes from 192.168.60.5: icmp_seq=40 ttl=63 time=0.058 ms
^C
--- 192.168.60.5 ping statistics ---
42 packets transmitted, 11 received, 73.8095% packet loss, time 41977ms
rtt min/avg/max/mdev = 0.058/0.167/0.256/0.073 ms
root@56f473f43aa8:/#
```

From here I understood that without the second rule, the first rule alone is not useful.

Here are my some observations:

- The second rule is needed for sure to achieve the limiting the number of packets that can pass through the firewall
- After implementing the second rule, initially as per the first rule, 5 packets are being transmitted immediately without any gap.
- Once those 5 packets are transmitted from the 6th packet, the packet loss starts and after nearly 6 seconds a packet is getting transmitted successfully.

For the better understanding, I modified the first rule a little bit that from 2nd packet itself the dropping starts, so that at exactly after the 11th successfully transmitted packet the time will be 60 seconds

```
root@f863f0cb47dd:/# iptables -A FORWARD -s 10.9.0.5 -m limit \--limit 10/minute --limit-burst 1 -j ACCEPT
root@f863f0cb47dd:/# iptables -A FORWARD -s 10.9.0.5 -j DROP
root@f863f0cb47dd:/# iptables  -L -n
Chain INPUT (policy ACCEPT)
target     prot opt source               destination

Chain FORWARD (policy ACCEPT)
target     prot opt source               destination
ACCEPT     all  --  10.9.0.5            0.0.0.0/0            limit: avg 10/min burst 1
DROP       all  --  10.9.0.5            0.0.0.0/0

Chain OUTPUT (policy ACCEPT)
target     prot opt source               destination
root@f863f0cb47dd:/#
```

Testing the connection by ping 192.168.60.5 from 10.9.0.5

```
root@56f473f43aa8:/# ping 192.168.60.5
PING 192.168.60.5 (192.168.60.5) 56(84) bytes of data.
64 bytes from 192.168.60.5: icmp_seq=1 ttl=63 time=0.132 ms
64 bytes from 192.168.60.5: icmp_seq=7 ttl=63 time=0.245 ms
64 bytes from 192.168.60.5: icmp_seq=13 ttl=63 time=0.255 ms
64 bytes from 192.168.60.5: icmp_seq=19 ttl=63 time=0.248 ms
64 bytes from 192.168.60.5: icmp_seq=25 ttl=63 time=0.134 ms
64 bytes from 192.168.60.5: icmp_seq=31 ttl=63 time=0.296 ms
64 bytes from 192.168.60.5: icmp_seq=37 ttl=63 time=0.147 ms
64 bytes from 192.168.60.5: icmp_seq=43 ttl=63 time=0.250 ms
64 bytes from 192.168.60.5: icmp_seq=49 ttl=63 time=0.095 ms
64 bytes from 192.168.60.5: icmp_seq=55 ttl=63 time=0.272 ms
64 bytes from 192.168.60.5: icmp_seq=61 ttl=63 time=0.276 ms
^C
--- 192.168.60.5 ping statistics ---
61 packets transmitted, 11 received, 81.9672% packet loss, time 61433ms
rtt min/avg/max/mdev = 0.095/0.213/0.296/0.067 ms
root@56f473f43aa8:/#
```

Restore the filter table to its original state

```
root@f863f0cb47dd:/# iptables -F
root@f863f0cb47dd:/# iptables -L -n
Chain INPUT (policy ACCEPT)
target     prot opt source               destination

Chain FORWARD (policy ACCEPT)
target     prot opt source               destination

Chain OUTPUT (policy ACCEPT)
target     prot opt source               destination
```

Task 5: Load Balancing

Starting all the internal hosts

host1-192.168.60.5

```
[03/30/23]seed@VM:~/.../Labsetup$ docksh host1-192.168.60.5
root@06d66a6c664d:/#
```

host2-192.168.60.6

```
[03/30/23]seed@VM:~/.../Labsetup$ docksh host2-192.168.60.6
root@ff64c9ae7dae:/#
```

host3-192.168.60.7

```
[03/30/23]seed@VM:~/.../Labsetup$ docksh host3-192.168.60.7
root@f78e15e8c54a:/#
```

Starting servers on each hosts

host1-192.168.60.5

```
[03/30/23]seed@VM:~/.../Labsetup$ docksh host1-192.168.60.5
root@06d66a6c664d:/# nc -luk 8080
```

Starting NetCat listener on UDP protocol on port 8080 that Keep the listener running after a client disconnects, allowing multiple connections to be made to the same port on the internal host1 (192.168.60.5)

host2-192.168.60.6

```
[03/30/23]seed@VM:~/.../Labsetup$ docksh host2-192.168.60.6
root@ff64c9ae7dae:/# nc -luk 8080
```

Starting NetCat listener on UDP protocol on port 8080 that Keep the listener running after a client disconnects, allowing multiple connections to be made to the same port on the internal host2 (192.168.60.6)

host3-192.168.60.7

```
[03/30/23]seed@VM:~/.../Labsetup$ docksh host3-192.168.60.7
root@f78e15e8c54a:/# nc -luk 8080
```

Starting NetCat listener on UDP protocol on port 8080 that Keep the listener running after a client disconnects, allowing multiple connections to be made to the same port on the internal host3 (192.168.60.7)

Using the nth mode (round-robin)

Implementing the rule

```
root@f863f0cb47dd:/# iptables -t nat -A PREROUTING -p udp --dport 8080 -m statistic --mode nth --every 3 --packet 0 -j DNAT --to-destination
192.168.60.5:8080
```

iptables -t nat -A PREROUTING -p udp --dport 8080 -m statistic --mode nth --every 3 --packet 0 -j DNAT --to-destination 192.168.60.5:8080

This rule implements that

- This rule adds a rule to the NAT table in order to modify the destination address of incoming UDP packets with a destination port of 8080, based on a statistic.
- The "nth" statistic is used with an "every" value of 3 and a "packet" value of 0, which means that the first packet matching this rule will have its destination address changed to 192.168.60.5:8080, the second packet will be unaffected, and the third packet will also have its destination address changed, and so on in a repeating pattern.
- The DNAT target is used to specify the new destination address for the packet

Testing the connection

Sending packets from 10.9.0.5

```
[03/30/23]seed@VM:~/.../Labsetup$ docksh hostA-10.9.0.5
root@56f473f43aa8:/# echo hello | nc -u 10.9.0.11 8080
^C
root@56f473f43aa8:/#
```

From the external machine (10.9.0.5) sending ‘hello’ message over UDP to the router (10.9.0.11) on port 8080 to reach the internal host 1 (192.168.60.5)

The packets received on 192.168.60.5

```
[03/30/23]seed@VM:~/.../Labsetup$ docksh host1-192.168.60.5
root@06d66a6c664d:/# nc -luk 8080
hello
```

At the internal host 1 (192.168.60.5), the ‘hello’ message received over UDP

On the remaining 2 hosts, no packets received
192.168.60.6

```
[03/30/23]seed@VM:~/.../Labsetup$ docksh host2-192.168.60.6
root@ff64c9ae7dae:/# nc -luk 8080
```

192.168.60.7

```
[03/30/23]seed@VM:~/.../Labsetup$ docksh host3-192.168.60.7
root@f78e15e8c54a:/# nc -luk 8080
```

Add more rules to the router container, so all the three internal hosts get the equal number of packets

```
root@f863f0cb47dd:/# iptables -t nat -A PREROUTING -p udp --dport 8080 -m statistic --mode nth --every 3 --packet 0 -j DNAT --to-destination
192.168.60.5:8080
root@f863f0cb47dd:/# iptables -t nat -A PREROUTING -p udp --dport 8080 -m statistic --mode nth --every 3 --packet 1 -j DNAT --to-destination
192.168.60.6:8080
root@f863f0cb47dd:/# iptables -t nat -A PREROUTING -p udp --dport 8080 -m statistic --mode nth --every 3 --packet 2 -j DNAT --to-destination
192.168.60.7:8080
root@f863f0cb47dd:/# iptables -t nat -L -n
Chain PREROUTING (policy ACCEPT)
target     prot opt source               destination
DNAT       udp  --  0.0.0.0/0            0.0.0.0/0            udp dpt:8080 statistic mode nth every 3 to:192.168.60.5:8080
DNAT       udp  --  0.0.0.0/0            0.0.0.0/0            udp dpt:8080 statistic mode nth every 3 packet 1 to:192.168.60.6:8080
DNAT       udp  --  0.0.0.0/0            0.0.0.0/0            udp dpt:8080 statistic mode nth every 3 packet 2 to:192.168.60.7:8080

Chain INPUT (policy ACCEPT)
target     prot opt source               destination

Chain OUTPUT (policy ACCEPT)
target     prot opt source               destination
DOCKER_OUTPUT  tcp  --  0.0.0.0/0            0.0.0.0/0            multiport dports 53
DOCKER_OUTPUT  udp  --  0.0.0.0/0            0.0.0.0/0            multiport dports 53

Chain POSTROUTING (policy ACCEPT)
target     prot opt source               destination
DOCKER_POSTROUTING  all  --  127.0.0.11           0.0.0.0/0

Chain DOCKER_OUTPUT (2 references)
target     prot opt source               destination
DNAT       all  --  0.0.0.0/0            0.0.0.0/0            to:127.0.0.11

Chain DOCKER_POSTROUTING (1 references)
target     prot opt source               destination
SNAT       tcp  --  127.0.0.11           0.0.0.0/0            tcp spt:33815 to::53
SNAT       udp  --  127.0.0.11           0.0.0.0/0            udp spt:41425 to::53
root@f863f0cb47dd:/#
```

Here i implemented 3 rules for each internal hosts

- For internal host 1, this iptables rule will match incoming UDP packets with destination port 8080 and use the statistic module to apply the load balancing policy. Specifically, the --mode nth --every 3 --packet 0 option means that every third packet will be redirected to the IP address and port specified by --to-destination (in this case, 192.168.60.5:8080).
- For internal host 2, the rule forwards every third UDP packet with a destination port of 8080 to the IP address 192.168.60.5:8080, and every third packet with a destination port of 8080 and packet number 1 to the IP address 192.168.60.6:8080
- For internal host 3, the rule forwards every third UDP packet with a destination port of 8080 to the IP address 192.168.60.5:8080, and every third packet with a destination port of 8080 and packet number 2 to the IP address 192.168.60.6:8080

Trying to send equal number of packets to the internal hosts

From 10.9.0.5

```
[03/30/23]seed@VM:~/.../Labsetup$ docksh hostA-10.9.0.5
root@56f473f43aa8:/# echo hello | nc -u 10.9.0.11 8080
^C
root@56f473f43aa8:/# for i in {1..10}; do echo "hello $i" | nc -u 10.9.0.11 8080; done
^C
root@56f473f43aa8:/# for i in {1..10}; do echo "hello $i" | nc -u 10.9.0.11 8080; done
^C
root@56f473f43aa8:/# for i in {1..10}; do echo "hello $i" | nc -u 10.9.0.11 8080; done
^C
root@56f473f43aa8:/#
```

Trying to send the multiple ‘hello’ packets, to the router for load balancing on the internal servers

At 192.168.60.5

```
[03/30/23]seed@VM:~/.../Labsetup$ docksh host1-192.168.60.5
root@06d66a6c664d:/# nc -luk 8080
hello
hello 1
^C
root@06d66a6c664d:/# nc -luk 8080
```

Out of multiple attempts to send the packets only few times the packet is forwarding to the internal host 1 (192.168.60.5)

At 192.168.60.6

```
[03/30/23]seed@VM:~/.../Labsetup$ docksh host2-192.168.60.6
root@ff64c9ae7dae:/# nc -luk 8080
hello 1
^C
root@ff64c9ae7dae:/# nc -luk 8080
hello 1
```

Out of multiple attempts to send the packets only few times the packet is forwarding to the internal host 2 (192.168.60.6)

At 192.168.60.7

```
[03/30/23]seed@VM:~/.../Labsetup$ docksh host3-192.168.60.7
root@f78e15e8c54a:/# nc -luk 8080
hello 1
^C
root@f78e15e8c54a:/# nc -luk 8080
```

Out of multiple attempts to send the packets only few times the packet is forwarding to the internal host 3 (192.168.60.7)

Restore the filter table to its original state

```
root@f863f0cb47dd:/# iptables -t nat -D PREROUTING 1
root@f863f0cb47dd:/# iptables -t nat -L -n
Chain PREROUTING (policy ACCEPT)
target     prot opt source               destination

Chain INPUT (policy ACCEPT)
target     prot opt source               destination

Chain OUTPUT (policy ACCEPT)
target     prot opt source               destination
DOCKER_OUTPUT  tcp  --  0.0.0.0/0            0.0.0.0/0            multiport dports 53
DOCKER_OUTPUT  udp  --  0.0.0.0/0            0.0.0.0/0            multiport dports 53

Chain POSTROUTING (policy ACCEPT)
target     prot opt source               destination
DOCKER_POSTROUTING  all  --  127.0.0.11           0.0.0.0/0

Chain DOCKER_OUTPUT (2 references)
target     prot opt source               destination
DNAT       all  --  0.0.0.0/0            0.0.0.0/0            to:127.0.0.11

Chain DOCKER_POSTROUTING (1 references)
target     prot opt source               destination
SNAT       tcp  --  127.0.0.11           0.0.0.0/0            tcp spt:33815 to::53
SNAT       udp  --  127.0.0.11           0.0.0.0/0            udp spt:41425 to::53
root@f863f0cb47dd:/#
```

Observation

- **There is one assumption why the message does not reach all the servers at a time because of only one external host, if we have multiple external hosts then the load balancing might work as expected.**

Using the random mode

Writing the rules

```
[03/30/23]seed@VM:~/.../Labsetup$ docksh seed-router
root@f863f0cb47dd:/# iptables -t nat -A PREROUTING -p udp --dport 8080 -m statistic --mode random --probability 0.333 -j DNAT --to-destinatio
n 192.168.60.5:8080
root@f863f0cb47dd:/# iptables -t nat -A PREROUTING -p udp --dport 8080 -m statistic --mode random --probability 0.6 -j DNAT --to-destination
192.168.60.6:8080
root@f863f0cb47dd:/# iptables -t nat -A PREROUTING -p udp --dport 8080 -m statistic --mode random --probability 0.9 -j DNAT --to-destination
192.168.60.7:8080
root@f863f0cb47dd:/# iptables -t nat -L -n
Chain PREROUTING (policy ACCEPT)
target     prot opt source               destination
DNAT       udp  --  0.0.0.0/0            0.0.0.0/0            udp dpt:8080 statistic mode random probability 0.33300000010 to:192.168.60.5:80
80
DNAT       udp  --  0.0.0.0/0            0.0.0.0/0            udp dpt:8080 statistic mode random probability 0.60000000009 to:192.168.60.6:80
80
DNAT       udp  --  0.0.0.0/0            0.0.0.0/0            udp dpt:8080 statistic mode random probability 0.89999999991 to:192.168.60.7:80
80

Chain INPUT (policy ACCEPT)
target     prot opt source               destination

Chain OUTPUT (policy ACCEPT)
target     prot opt source               destination
DOCKER_OUTPUT  tcp  --  0.0.0.0/0            0.0.0.0/0            multiport dports 53
DOCKER_OUTPUT  udp  --  0.0.0.0/0            0.0.0.0/0            multiport dports 53

Chain POSTROUTING (policy ACCEPT)
target     prot opt source               destination
DOCKER_POSTROUTING  all  --  127.0.0.11           0.0.0.0/0

Chain DOCKER_OUTPUT (2 references)
target     prot opt source               destination
DNAT       all  --  0.0.0.0/0            0.0.0.0/0            to:127.0.0.11

Chain DOCKER_POSTROUTING (1 references)
```

Here I implemented 3 rules for the 3 internal servers

- The first rule adds a rule to the NAT table's PREROUTING chain that performs random load balancing for UDP packets with a destination port of 8080.there is a 33.3% chance that a packet with a destination port of 8080 will be load-balanced to 192.168.60.5:8080, while the remaining packets will be load-balanced using the other rules in the PREROUTING chain
- The second rule adds a rule to the NAT table's PREROUTING chain that performs random load balancing for UDP packets with a destination port of 8080.there is a 60% chance that a packet with a destination port of 8080 will be load-balanced to 192.168.60.6:8080, while the remaining packets will be load-balanced using the other rules in the PREROUTING chain
- The third rule adds a rule to the NAT table's PREROUTING chain that performs random load balancing for UDP packets with a destination port of 8080.there is a 90% chance that a packet with a destination port of 8080 will be load-balanced to 192.168.60.7:8080, while the remaining packets will be load-balanced using the other rules in the PREROUTING chain

Trying to send packets to the internal hosts

```
[03/30/23]seed@VM:~/.../Labsetup$ docksh hostA-10.9.0.5
root@56f473f43aa8:/# for i in {1..10}; do echo "hello $i" | nc -u 10.9.0.11 8080; done
^C
root@56f473f43aa8:/# for i in {1..10}; do echo "hello $i" | nc -u 10.9.0.11 8080; done
^C
root@56f473f43aa8:/# for i in {1..10}; do echo "hello $i" | nc -u 10.9.0.11 8080; done
^C
root@56f473f43aa8:/# for i in {1..10}; do echo "hello $i" | nc -u 10.9.0.11 8080; done
^C
root@56f473f43aa8:/# for i in {1..10}; do echo "hello $i" | nc -u 10.9.0.11 8080; done
^C
root@56f473f43aa8:/# for i in {1..10}; do echo "hello $i" | nc -u 10.9.0.11 8080; done
^C
root@56f473f43aa8:/# for i in {1..10}; do echo "hello $i" | nc -u 10.9.0.11 8080; done
^C
root@56f473f43aa8:/# for i in {1..10}; do echo "hello $i" | nc -u 10.9.0.11 8080; done
^C
root@56f473f43aa8:/#
```

Trying to send the multiple 'hello' packets, to the router for load balancing on the internal servers. No matter if i'm using the for loop to send 10 packets at a time, it is accepting only 1 packet per

At 192.168.60.5

```
[03/30/23]seed@VM:~/.../Labsetup$ docksh host1-192.168.60.5
root@06d66a6c664d:/# nc -luk 8080
hello 1
hello 1
hello 1
```

Out of multiple attempts to send the packets only few times the packet is forwarding to the internal host 1 (192.168.60.5)

At 192.168.60.6

```
[03/30/23]seed@VM:~/.../Labsetup$ docksh host2-192.168.60.6
root@ff64c9ae7dae:/# nc -luk 8080
hello 1
hello 1
hello 1
hello 1
```

Out of multiple attempts to send the packets only few times the packet is forwarding to the internal host 2 (192.168.60.6)

At 192.168.60.7

```
[03/30/23]seed@VM:~/.../Labsetup$ docksh host3-192.168.60.7
root@f78e15e8c54a:/# nc -luk 8080
hello 2
```

Out of multiple attempts to send the packets only few times the packet is forwarding to the internal host 3 (192.168.60.7)

REFERENCES

(n.d.). The User Datagram Protocol (UDP). Retrieved March 30, 2023, from https://erg.abdn.ac.uk/users/gorry/course/inet-pages/udp.html

(2023, March 14). huihoo. Retrieved March 30, 2023, from https://docs.huihoo.com/doxygen/linux/kernel/3.7/uapi_2linux_2icmp_8h_source.html

command line - Where is .bashrc? (n.d.). Ask Ubuntu. Retrieved March 30, 2023, from https://askubuntu.com/questions/127056/where-is-bashrc

ICMP (Internet Control Message Protocol). (n.d.). NetworkLessons.com. Retrieved March 30, 2023, from https://networklessons.com/cisco/ccie-routing-switching-written/icmp-internet-control-message-protocol

Internet Control Message Protocol (ICMP) Parameters. (2020, September 25). Internet Assigned Numbers Authority. Retrieved March 30, 2023, from https://www.iana.org/assignments/icmp-parameters/icmp-parameters.xhtml

iptables(8) - Linux man page. (n.d.). iptables(8) - Linux man page. Retrieved March 30, 2023, from https://linux.die.net/man/8/iptables

linux kernel - How long does conntrack remember a connection? - Unix & Linux Stack Exchange. (2019, June 11). Unix Stack Exchange. Retrieved March 30, 2023, from https://unix.stackexchange.com/questions/524295/how-long-does-conntrack-remember-a-connection

nc (Netcat) Command: Syntax, Command Options, & Examples. (2022, May 24). phoenixNAP. Retrieved March 30, 2023, from https://phoenixnap.com/kb/nc-command

Stender, A. (n.d.). *Connection tracking (conntrack) - Part 3: State and Examples*. Thermalcircle.de. Retrieved March 30, 2023, from https://thermalcircle.de/doku.php?id=blog:linux:connection_tracking_3_state_and_examples

(n.d.). ntohs(3) - Linux man page. Retrieved March 30, 2023, from https://linux.die.net/man/3/ntohs